

Resumen

1. Introducción	6
Algunas definiciones	6
DMBS	6
System Catalog	6
Arquitectura e Independencia	7
Esquemas	7
Independencia	7
2. Modelización	8
Modelo Entidad - Relación (MER)	8
Entidades	8
Tipos de Entidades	8
Atributos	8
Tipos de atributos	8
Interrelaciones	8
Agregación	9
Modelo Lógico Relacional (MR)	9
3. Lenguajes de Consulta	10
AR (Álgebra Relacional)	10
Operaciones Unarias	10
Operaciones Binarias	10
Union, Intersection, Minus	10
Producto Cartesiano	11
Join	11
Natural Join	11
División	12
CRT (Cálculo Relacional de Tuplas)	12
Expresión general	12
Expresiones Seguras	12
4. Ehrenfeucht - Fraisse	13
El Juego	13
Quantifier Rank	13
Teorema de Ehrenfeucht-Fraisse	13
Aplicación: Demostración de Inexpresividad de propiedad usando LPO	13
5. Normalización	14
Claves	14
Dependencias funcionales	14
Tipos de DFs	14

Formas Normales	14
Primer Forma Normal (1FN)	15
Segunda Forma Normal (2FN)	15
Tercera Forma Normal (3FN)	15
Descomposición Boyce-Codd (BCFN)	15
BCFN vs 3FN	15
Inferencia	15
Reglas de Inferencia (Axiomas de Armstrong)	15
Conjunto Minimal de DFs	16
Descomposición de Relaciones	16
6. Transacciones	17
Propiedades ACID	17
Problemas de Control de Concurrencia	17
Lost Update	17
Dirty Read	17
Unrepeatable Read	17
Incorrect Summary	18
Historias	18
Tipos de Historias	18
Verificación de Serializabilidad	18
Paradigma Pesimista	19
Tipos de Locks	19
Two Phase Locking (2PL)	19
Variantes de 2PL	19
Paradigma Optimista	19
Método: Timestamping	20
Reglas para el planificador	20
Método: Timestamping Multiversión	21
Paradigma Optimista vs Paradigma Pesimista	21
Similitudes	21
Diferencias	21
Recuperación	22
Log del Sistema	22
Políticas de Recuperación	22
Sin Checkpoint	22
Undo / Checkpoint Quiescente	23
Checkpoint No Quiescente	24
7. Optimización	25
El Query Optimizer	25
Relaciones	25

Almacenamiento físico (Archivos)	26
Índices	26
Propiedades	26
Índices Clustered vs Unclustered	26
Índices Densos vs No Densos	27
Índices Primarios vs Secundarios	27
Representaciones	27
Índice Árbol B+	27
Índice basado en hash estático	27
Cuadro resumen	28
File Scan e Index Scan	28
Join	28
Block Nested Loop Join(BNLJ)	28
Index Nested Loop Join(INLJ)	29
Sort Merge Join(SMJ)	29
Pipeline & Materialize	29
Optimizaciones Algebraicas	29
Heurísticas	30
8. NoSQL	30
Propiedades BASE	31
Modelo Relacional vs NoSQL	31
Taxonomías	31
Clave - Valor	31
Column Store	32
Column Family Store	32
Documentos	32
Gráfos	32
Herramientas / Operaciones	33
Map - Reduce	33
Sharding	33
Réplicas	33
Método de implementación: Master - Slave	34
Método de implementación: P2P (Peer To Peer)	34
Sharding + Réplicas	34
9. Bases de Datos Distribuídas (DDB)	35
Fragmentación	35
Replicación	35
Replicación total	36
Replicación parcial	36
Control de Concurrencia	36

Copia Distinguida	36
Votación	36
Recovery	37
Transacciones	37
2PC	37
3PC	37
Procesamiento de Queries	38
Tipos de DDB	38
Paralelas vs Distribuídas	38
Catálogo	39
Esquemas de Administración de Catálogos de DDBMS	39
Teorema CAP	39
10. Extensiones y Otras Bases	41
ORM (Mapeo Objeto - Relacional)	41
XML (Extensible Markup Language)	41
Stream Databases	42
Base de datos espacial	42
Base de datos móviles	42
RDF (Resource Description Format) Management Systems	42
11. Data Mining	43
Técnicas	43
Supervisadas	43
Redes neuronales	43
Árboles de decisión	43
Regresión lineal	43
No supervisadas	44
Clustering	44
Reglas de Asociación	44
12. Datawarehouse	45
Datos Operacionales vs Datawarehouse	45
Explotación del Datawarehouse / ETL	45
Modelado de datos	46
Técnicas	46
Modelo E - R	46
Modelo Dimensional	46
Cubo OLAP	47
Operaciones	47
Modelos básicos dimensionales	48
13. Long Duration Transactions	49

14. Open Data	50
15. Administrador de Datos	51
Diferencia entre DBA y administrador de datos	51
Tareas de un administrador de datos	51
Bibliografía	52

1. Introducción

Algunas definiciones

Base de datos: Conjunto de datos relacionado con un significado inherente (un conjunto aleatorio de datos no es considerado una base de datos). Una base de datos es diseñada y construida con un propósito específico.

Dato: Hecho conocido que puede ser registrado y tiene un significado implícito.

DML (Data Manipulation Language): Lenguaje usado en los DBMS para obtener, agregar o modificar datos. Algunos ejemplos: *SELECT*, *INSERT*, *DELETE*.

DDL (Data Definition Language): Lenguaje usado para alterar el esquema de la base de datos. Algunos ejemplos: *CREATE*, *ALTER*, *DROP*, *RENAME*

DMBS

DBMS (database management system): Herramienta para crear y manejar largas cantidades de datos de forma eficiente. Se espera que un DBMS:

1. Permita a usuarios crear nuevas bases de datos y especificar sus esquemas.
2. Permita a usuarios realizar consultas respecto a los datos. También debe permitirles modificar los datos.
3. Almacene grandes cantidades de datos por un largo período de tiempo, permitiendo acceso eficiente a las consultas y modificaciones.
4. Garantice la durabilidad de los datos, al permitir la recuperación de la base en caso de fallas, errores o mal uso intencional.
5. Controle el acceso a los datos de muchos usuarios al mismo tiempo, sin que sucedan interacciones inesperadas entre ellos y garantizando que las acciones sobre los datos se realizan de forma completa.

Algunos componentes de una DBMS:

- Recovery Manager: encargado de restaurar la base de datos a un estado consistente en caso de falla. Utiliza el *log*.
- Optimizador de consultas: encargado de armar un plan de ejecución eficiente de una query. Utiliza el *system catalog*.

System Catalog

Definición: El system catalog es el lugar donde un RDBMS guarda los metadatos del esquema. Los metadatos pueden incluir:

- Información sobre las tablas y columnas
 - Datos estadísticos como:
 - Tamaño de archivos y factor de bloqueo.
 - Cantidad de tuplas de la relación.
 - Cantidad de bloques que ocupa la relación.
 - Cantidad de valores distintos de una columna.
- Views
- Índices

- Usuarios y grupos de usuario.
- Triggers
- Funciones de agregación de finitas por el usuario.

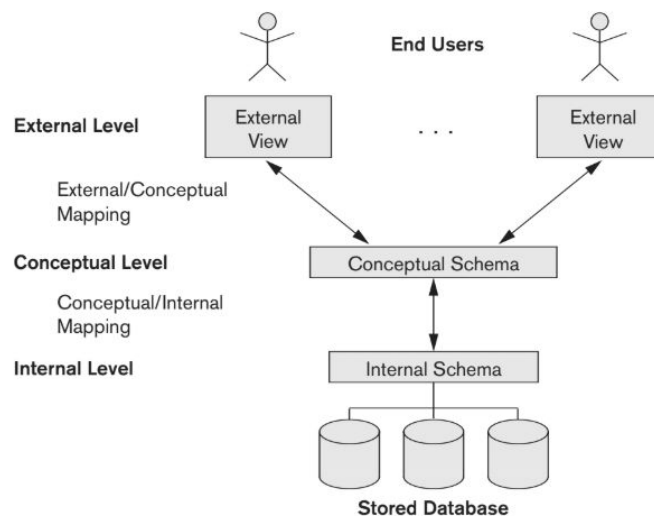
Algunos usos que se le da al system catalog:

- Obtener el esquema de una tabla a la hora de verificar una query.
- Obtener la selectividad esperada de un atributo a la hora de optimizar una query.
- Obtener los permisos de un usuario a la hora de verificar un acceso.

Arquitectura e Independencia

Esquemas

1. **Interno:** Describe el almacenamiento físico de la estructura de la DB.
2. **Conceptual:** El nivel conceptual tiene un *esquema conceptual*, el cual describe la estructura de toda la base de datos. El *esquema conceptual* esconde los detalles de las estructuras físicas de almacenamiento y se concentra en describir entidades, tipos de datos, operaciones de usuarios y restricciones.
3. **Externo (o de vista):** Este nivel incluye a un número de esquemas externos o vistas de usuarios. Cada esquema externo describe la parte de la base de datos que le interesa a un grupo particular de usuarios, y esconde el resto de la base para ese grupo.



Independencia

Independencia lógica: La independencia lógica es la capacidad de cambiar el esquema conceptual sin la necesidad de cambiar los esquema externos. Podríamos cambiar el esquema conceptual para expandir la base de datos (al agregar un nuevo tipo de registro), para cambiar las restricciones o reducir la base (quitar un tipo de registro). Por lo general, este tipo de independencia es difícil de lograr.

Independencia física: La independencia física es la capacidad de poder cambiar el esquema interno sin tener que cambiar el esquema conceptual (y por lo tanto tampoco los esquema externos). Algunos potenciales cambios al esquema interno podrían ser la organización de los archivos, o el agregado de un índice.

2. Modelización

En líneas generales, los grandes pasos a realizar para lograr un diseño de la Base de Datos Relacional con esta metodología son:

Requerimientos <-> MER <-> MR <-> Normalización <-> Diseño Físico <-> BD

Modelo Entidad - Relación (MER)

El Modelo Entidad-Relación es una herramienta que permite realizar una abstracción o modelo de alguna situación de interés presente en el mundo real. El MER se realizará utilizando la técnica Diagramas de Entidad Relación (DER).

El MER se basa en: entidades, atributos e interrelaciones.

Entidades

Todo objeto o concepto del cual queremos registrar información constituye una entidad. Las entidades representan conjuntos de elementos.

Tipos de Entidades

Las entidades fuertes son aquellas que tienen una existencia independientemente de cualquier otra entidad, se identifican sólo por atributos propios.

Las entidades débiles son aquellas que derivan su existencia de otra entidad y necesitan la identificación de dicha entidad para distinguirse de otras.

Atributos

Los atributos son propiedades descriptivas de las entidades. Constituyen la información concreta que queremos mantener para cada elemento de una entidad.

Tipos de atributos

Compuestos vs Simples (atómicos):

Los compuestos pueden ser divididos en sub-partes que representan atributos con significados independientes. Por ejemplo, dirección puede ser dividida en calle, ciudad, estado, etc. Estas sub-partes pueden estar divididas también incluso. Pueden utilizarse en los casos que se requiera el atributo como un conjunto de sub-partes o como entero.

Los atributos atómicos son indivisibles.

Single-Valued vs Multivalued:

Los single-valued tienen un solo valor para un atributo. Ejemplo, edad.

Los multivalued pueden tener más de un valor al mismo tiempo.

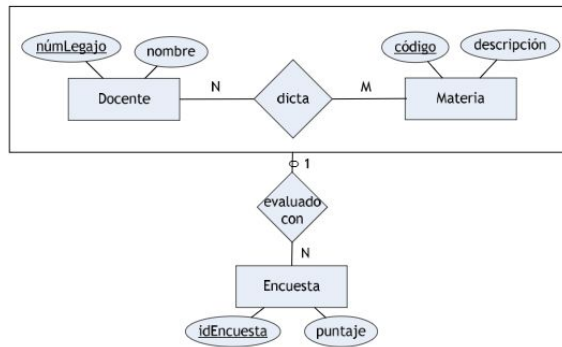
Por ejemplo: los idiomas que conoce una persona.

Interrelaciones

Las diferentes entidades no están aisladas en el dominio del problema, muchas de ellas van a estar vinculadas entre sí. A esta vinculación la llamamos interrelación. En una interrelación pueden participar una entidad o muchas entidades.

Agregación

La agregación es una abstracción en la cual una interrelación (junto con sus entidades vinculadas) es tratada como una entidad de alto nivel y puede participar de interrelaciones. Por supuesto, las entidades vinculadas a una agregación también pueden tener sus propias interrelaciones individualmente.



Modelo Lógico Relacional (MR)

En este modelo:

- Una base de datos es un conjunto de *relaciones*.
- Una relación puede pensarse como una tabla, con filas y columnas.
- Cada fila, a la que denominaremos *tupla* está formada por un conjunto de valores de datos relacionados que representan un hecho de la realidad.
- Cada *columna* de la tabla representa un atributo, asociado a un conjunto de valores posibles que puede tomar. A este conjunto lo llamamos dominio del atributo.

3. Lenguajes de Consulta

AR (Álgebra Relacional)

Definición: El AR es un lenguaje formal utilizado en el modelo relacional que permite a usuarios especificar consultas sobre instancias de relaciones. El resultado de una consulta es una nueva relación.

Técnica: Procedural (a diferencia del CRT que es de tipo declarativo)

Operaciones Unarias

Operación	Select	Project	Rename
Descripción	Selecciona un subconjuntos de tuplas de una relación que satisface cierta condición. Genera una <i>partición horizontal</i> de la relación.	Selecciona un subconjunto de columnas de una relación. Genera una <i>partición vertical</i> de la relación.	Asigna nombre a atributos relación resultado
Ejemplo	$\sigma_{\text{Sexo}=F}(\text{EMPLEADO})$	$\pi_{\text{Sexo}}(\text{EMPLEADO})$	$\text{EMP}(\text{id}, \text{Ingreso}) \leftarrow \pi_{\text{DNI}, \text{Salario}}(\text{EMPLEADO})$
Propiedades	1. Conmutatividad: $\sigma_{c_1}(\sigma_{c_2}(R)) = \sigma_{c_2}(\sigma_{c_1}(R))$ 2. Cascada: $\sigma_{c_1}(\sigma_{c_2}(\dots \sigma_{c_n}(R))) = \sigma_{c_1 \text{ AND } c_2 \text{ AND } \dots \text{ AND } c_n}(R)$ 3. # atributos se preserva	1. NO conmutatividad 2. # tuplas se preserva (si los atributos proyectados son súper clave)	-
Ejemplo SQL	SELECT * FROM Rel WHERE Condición	SELECT DISTINCT Columnas FROM Rel	SELECT NRel.Col AS NCol FROM Rel AS NRel

Operaciones Binarias

Union, Intersection, Minus

Operación	Union	Intersection	Minus
Descripción	Relación que incluye todas las tuplas que están en R, S o en ambas relaciones a la vez. Duplicados son eliminados	Relación que incluye todas las tuplas que están a la vez en R y S.	Relación que incluye todas las tuplas que están R, pero no incluye aquellas que aparecen en S
Notación	$R \cup S$	$R \cap S$	$R - S$
Propiedades	1. Conmutatividad: $R \cup S = S \cup R$ 2. Asociatividad: $R \cup (S \cup T) = (R \cup S) \cup T$	1. Conmutatividad: $R \cap S = S \cap R$ 2. Asociatividad: $R \cap (S \cap T) = (R \cap S) \cap T$	Por lo general $R - S \neq S - R$
Ejemplo SQL	SELECT Columnas FROM Rel1 UNION SELECT Columnas FROM Rel2	SELECT Columnas FROM Rel1 INTERSECT SELECT Columnas FROM Rel2	SELECT Columnas FROM Rel1 EXCEPT SELECT Columnas FROM Rel2

Producto Cartesiano

Descripción: Produce una nueva relación que combina cada tupla de una relación con cada una de las tuplas de la otra relación.

Notación: $R \times S$

PERSONA		NACIONALIDADES	
Nombre	Nacionalidad	IDN	Detalle
Diego	AR	AR	Argentina
Laura	BR	BR	Brasilera
Marina	AR	CH	Chilena

• $RESULT \leftarrow PERSONA \times NACIONALIDADES$

Nombre	Nacionalidad	IDN	Detalle
Diego	AR	AR	Argentina
Diego	AR	BR	Brasilera
Diego	AR	CH	Chilena
Laura	BR	AR	Argentina
Laura	BR	BR	Brasilera
Laura	BR	CH	Chilena
Marina	AR	AR	Argentina
Marina	AR	BR	Brasilera
Marina	AR	CH	Chilena

Propiedades:

- **Grado:** Si $T = R \times S$ entonces $grado(T) = grado(R) + grado(S)$

Ejemplo SQL: `SELECT * FROM Rel1 CROSS JOIN Rel2;`

Join

Descripción: Permite combinar pares de tuplas relacionadas entre dos relaciones.

Notación: $R \bowtie_{\langle \text{condición} \rangle} S$

PERSONA		NACIONALIDADES	
Nombre	Nacionalidad	IDN	Detalle
Diego	AR	AR	Argentina
Laura	BR	BR	Brasilera
Marina	AR	CH	Chilena

• $RESULT \leftarrow PERSONA \bowtie_{Nacionalidad=IDN} NACIONALIDADES$

Nombre	Nacionalidad	IDN	Detalle
Diego	AR	AR	Argentina
Laura	BR	BR	Brasilera
Marina	AR	AR	Argentina

Propiedades:

- **NULL:** Tuplas cuyos atributos de JOIN son NULL o cuya condición es falsa no aparecen en el resultado.
- **Conmutatividad:** $R \bowtie_{\langle \text{condición} \rangle} S = S \bowtie_{\langle \text{condición} \rangle} R$ (solo cambia el orden de los atributos)
- **Grado:** Sea $T = R \bowtie_{\langle \text{condición} \rangle} S$, entonces $grado(T) = grado(R) + grado(S)$
- **# tuplas resultado:** La cantidad de tuplas resultantes puede ir de 0 a $\#R \times \#S$.

Natural Join

Descripción: Realiza el join entre campos de mismo nombre y deja sólo uno de los campos duplicados.

Notación: $R \bowtie S$

Requerimiento: Requiere que los atributos de join tengan el mismo nombre. Caso contrario, se debe hacer un *rename* previo.

División

Descripción: Retorna los valores de R que se encuentran emparejados con TODOS los valores de S. Requiere que atributos de $S \subseteq$ atributos de R. El resultado contiene atributos de R menos atributos de S

Notación: $R \div S$

ALUMNOS		MATERIAS_1	MATERIAS_2	MATERIAS_3
Nombre	Materia	Materia	Materia	Materia
Diego	BD	BD		BD
Diego	PLP		TLENG	PLP
Laura	BD			TLENG
Laura	PLP			
Laura	TLENG			
Marina	BD			
Marina	TLENG			
Santiago	BD			
Santiago	PLP			
Santiago	TLENG			

ALUMNOS \div MATERIAS_1		ALUMNOS \div MATERIAS_2	ALUMNOS \div MATERIAS_3
Nombre		Nombre	Nombre
Diego		Laura	Laura
Laura		Marina	Santiago
Marina		Santiago	
Santiago			

CRT (Cálculo Relacional de Tuplas)

Técnica: Declarativa. No existe una descripción de “en qué orden” es evaluada la consulta (no es procedural).

Expresión general

$\{ t \mid COND(t) \}$

donde:

- t es una variable de tipo tupla
- t es la *única variable libre* de la expresión
- $COND$ es una fórmula bien formada de CRT

El resultado es el conjunto de todas las tuplas t tal que evaluadas bajo la $COND(t)$ son verdaderas.

Dominio de una expresión: es el conjunto de valores que aparecen como valores constantes en la expresión o bien existen en cualquier tupla de las relaciones a las que se hace referencia en la expresión.

Expresiones Seguras

Una expresión segura en CRT es aquella que garantiza producir una cantidad finita de tuplas como resultado.

Por ejemplo, $\{ t \mid \neg (t \in EMPLEADO) \}$ es insegura dado que produce una cantidad infinita de tuplas.

Importante:

CRT restringido a expresiones seguras es equivalente en poder de expresividad al Álgebra Relacional.

4. Ehrenfeucht - Fraisse

Existen propiedades que no son expresables en Lógica de Primer Orden (LPO). Para demostrarlo, EF utiliza una demostración basada en teoría de juegos.

El Juego

- El juego tiene 2 jugadores, Spoiler y Duplicator y 2 grafos A y B.
- En cada ronda Spoiler debe elegir un nodo de uno de los grafos y luego Duplicator debe elegir un nodo del otro grafo.
- El objetivo de Spoiler es demostrar que los grafos son distinguibles en una sentencia de LPO mientras que Duplicator quiere mostrar que no lo son.
- Los grafos serán indistinguibles en n rondas si la función que describe los nodos elegidos es un isomorfismo parcial (mantiene adyacencia e igualdad).

Quantifier Rank

Definición: Sea f una fórmula de LPO. El quantifier rank de f , denotado por $qr(f)$, es la profundidad del anidamiento de los cuantificadores en f .

Formalmente: $qr(f)$ se define inductivamente así:

- Si f es una fórmula atómica, entonces $qr(f) = 0$
- Si f es de la forma $\neg g$, para una fórmula g , entonces $qr(f) = qr(g)$
- Si f es de la forma $(g \wedge h)$, o $(g \vee h)$, entonces $qr(f) = \max\{qr(g), qr(h)\}$
- Si f es de la forma $(\exists x g)$, o $(\forall x g)$, entonces $qr(f) = qr(g) + 1$

Teorema de Ehrenfeucht-Fraisse

Sean A, B dos grafos. Las siguientes afirmaciones son equivalentes:

- A y B satisfacen las mismas sentencias de *quantifier rank* n .
- $A \sim_n B$ (Duplicator gana la n -ésima movida del juego EF con tablero A y B).

Aplicación: Demostración de Inexpresividad de propiedad usando LPO

Dada una propiedad P , no existe una sentencia en lógica de primer orden que exprese P , sii, para cada número entero positivo n , es posible hallar 2 grafos A y B tal que:

- La propiedad P es verdadera en A
- La propiedad P es falsa en B
- $A \sim_n B$ (Duplicator gana la n -ésima movida del juego EF con tablero A y B)

5. Normalización

Claves

Súper Clave (SK): Una SK de R es un subconjunto de atributos $S \subseteq R$ con la propiedad de que no hay dos tuplas t_1, t_2 en un estado legal $r(R)$ que cumplan $t_1(S) = t_2(S)$.

Clave (K): Una clave K es una SK con la propiedad adicional de que al remover cualquier atributo de K , deja de ser SK. Es decir, K es una SK minimal.

Clave Candidata (CK): Si un esquema posee más de una clave, cada una se denomina clave candidata.

Clave Primaria (PK): Una CK designada arbitrariamente

Atributo primo: Un atributo $a \in R$ es primo si es parte de alguna CK.

Dependencias funcionales

Informalmente: Restricción entre dos conjuntos de atributos X e Y de una BD. Los valores que toman los atributos de Y dependen de los valores que tomen X .

Definición:

Sean X e Y dos conjuntos de atributos incluidos en R , la dependencia funcional indicada como $X \rightarrow Y$ especifica una restricción sobre las posibles tuplas que pueden conformar una instancia r de R . La restricción es:

Para cualquiera dos tuplas t_1, t_2 en r tal que $t_1[X] = t_2[X]$, se debe cumplir $t_1[Y] = t_2[Y]$.

Propiedades:

- Si X es CK de R , entonces $X \rightarrow Y \forall$ subconjunto de atributos Y de R .
- Si X es CK de R , entonces $X \rightarrow R$.

Observación: Dada una relación con su datos, no es posible determinar sus DFs a través de sus valores. Es necesario conocer el significado y relación que existe entre los atributos que la componen.

Tipos de DFs

DF Completa: Una DF $X \rightarrow Y$ es completa si al eliminar algún atributo A de X , la DF deja de existir.

DF Parcial: Una DF $X \rightarrow Y$ es parcial si es posible eliminar algún atributo A de X y la DF continúa existiendo.

DF Transitiva: Una DF $X \rightarrow Y$ en R es transitiva si existe un conjunto de atributos Z en R tal que:

- Z no es subconjunto de ninguna clave.
- $X \rightarrow Z$
- $Z \rightarrow Y$

DF Trivial: Una DF $A \rightarrow B$ es trivial si B es un subconjunto de atributos de A .

Formas Normales

Por si solas, las formas normales no garantizan un buen diseño de bases de datos. Para eso, el proceso de normalización debe garantizar la existencia de dos propiedades:

- **Lossless Join:** Garantiza que no se generen tuplas espurias a la hora de realizar un natural join.
 - Esta propiedad debe lograrse a cualquier costo
- **Conservación de Dependencias Funcionales:** Garantiza que cada dependencia funcional quede representada en una sola tabla.
 - Esta propiedad es deseable, pero en algunos casos es sacrificada

Primer Forma Normal (1FN)

Un esquema R está en 1FN si sus atributos incluyen solo valores atómicos. En 1FN se prohíbe tener tuplas o conjuntos como atributos de relación.

Segunda Forma Normal (2FN)

Un esquema R está en 2FN si todo atributo no primo (no es parte de ninguna CK) A de R depende funcionalmente de manera completa de la PK de R .

Tercera Forma Normal (3FN)

Una relación está en 3FN si para toda dependencia funcional $X \rightarrow Y$ en R no trivial vale una de las siguientes condiciones:

- X es una SK de R .
- Y es un atributo primo de R .

Observemos que una relación que viola la 3FN es aquella en la que ambas condiciones son falsas. Esto puede pasar cuando un atributo no primo es determinado funcionalmente por:

1. Otro atributo no primo, ó
2. Un subconjunto propio de una clave.

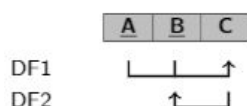
Descomposición Boyce-Codd (BCFN)

Una relación R está en BCFN si, para toda dependencia funcional no trivial $X \rightarrow A$ de R , X es SK de R .

BCFN vs 3FN

BCFN es más restrictiva que 3FN ya que BCFN no permite que A sea primo.

Visión esquemática de 3FN y no BCFN:



Inferencia

Definición: Una DF $X \rightarrow Y$ es inferida por un conjunto de DFs F de R si siempre que se satisface F , se cumple $X \rightarrow Y$.

Reglas de Inferencia (Axiomas de Armstrong)

- **R1 (regla reflexiva):** Si $Y \subseteq X$ entonces $X \rightarrow Y$
- **R2 (regla de incremento):** $\{X \rightarrow Y\} \Rightarrow XZ \rightarrow YZ$
- **R3 (regla transitiva):** $\{X \rightarrow Y, Y \rightarrow Z\} \Rightarrow X \rightarrow Z$

Clausura de F (F conjunto de DFs): Conjunto de todas las DFs de F más todas las DFs que puedan ser inferidas de F . Se denota como F^+ .

Clausura de X (X conjunto de atributos): Conjunto de atributos que son determinados por X basados en F . Se denota X^+ .

Equivalencia: Dados E y F conjuntos de DFs, F y E son equivalentes si F cubre a E y E cubre a F .

Cubrimiento: Dados E y F conjuntos de DFs, F cubre a E si $(\forall df \in E) df \in F^+$

Metodología: Para determinar si F cubre a G , calcular para cada DF $X \rightarrow Y$ de G , X^+ con respecto a F . Luego verificar si este X^+ incluye los atributos en Y .

Conjunto Minimal de DFs

Un cubrimiento minimal F' de un conjunto de dependencias funcionales F es un conjunto de dependencias funcionales que cumple:

- $F'^+ = F^+$
- El lado derecho de todas las dependencias en F' tiene un solo atributo.
- No hay atributos redundantes en el lado izquierdo.
- No hay dependencias funcionales redundantes.

Siempre hay un cubrimiento minimal. Se usa en el algoritmo de descomposición

Descomposición de Relaciones

Dada una descomposición de R en un subconjunto de esquemas, las siguiente propiedades son deseables:

- Preservación de atributos

$$\bigcup_{i=1}^m R_i = R$$

- Preservación de DFs
 - Si $X \rightarrow Y$ en F , es deseable que o bien aparezca en algún esquema R_i o que pueda ser inferida de las DFs de algún esquema R_i .
- Lossless Join / SPI (Sin Pérdida de Información)
 - El cumplimiento de esta propiedad no permite la generación de tuplas espúreas cuando se realiza un NATURAL JOIN entre las relaciones resultantes de una descomposición.

6. Transacciones

Definición: Una transacción es un conjunto de instrucciones que se ejecutan formando una unidad lógica de procesamiento. Una transacción puede incluir uno o más accesos a la BD a través del uso de diversas operaciones (inserción, eliminación, modificación, etc). Las cuatro operaciones básicas de una transacción son: read, write, commit y abort.

Propiedades ACID

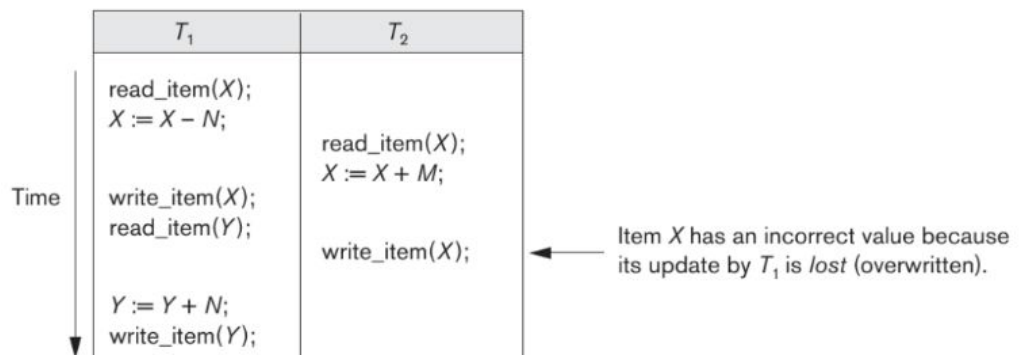
Las transacciones deben cumplir las propiedades ACID:

- **Atomicity:** una transacción debe ejecutarse completa o no ejecutarse del todo.
- **Consistency:** una transacción debe tomar una base de datos en estado válido y dejarlo en un estado válido.
- **Isolation:** ejecuciones concurrentes de transacciones deben arrojar el mismo resultado que si se hubieran ejecutado esas transacciones linealmente.
- **Durability:** una vez que una transacción commitea, los cambios que realiza son permanentes y no deben ser perdidos aún en el caso de fallas (eléctricas, físicas, lógicas, etc).

Problemas de Control de Concurrency

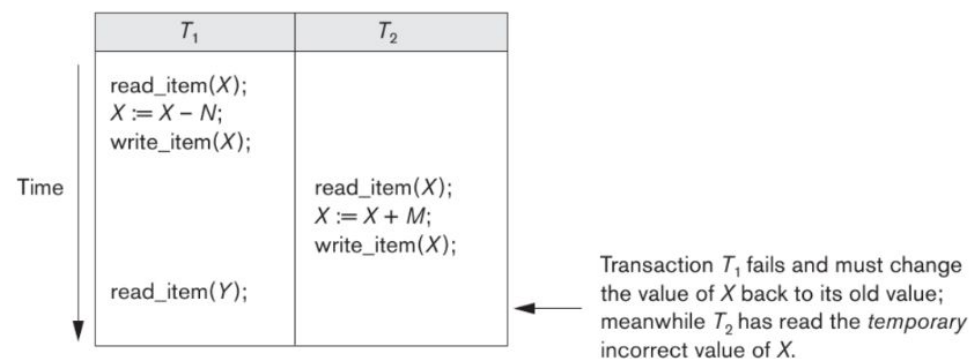
Lost Update

T_2 lee el valor de X previo a que T_1 lo modifique en la BD.



Dirty Read

T_2 lee el valor temporal de X , el cual no va a ser grabado en la BD debido a un fallo en medio de la transacción T_1 .



Unrepeatable Read

T lee el mismo ítem dos veces, pero el ítem es modificado por otra transacción T' entre ambas lecturas. Así, T recibe diferentes valores para estas dos lecturas del mismo ítem.

Incorrect Summary

T_3 lee el valor de X después de que N asientos han sido sustraídos por T_1 , pero lee el valor de Y antes de que esos N asientos hayan sido adicionados por T_1 .

T_1	T_3
$\text{read_item}(X);$ $X := X - N;$ $\text{write_item}(X);$	$\text{sum} := 0;$ $\text{read_item}(A);$ $\text{sum} := \text{sum} + A;$ \vdots
$\text{read_item}(Y);$ $Y := Y + N;$ $\text{write_item}(Y);$	$\text{read_item}(X);$ $\text{sum} := \text{sum} + X;$ $\text{read_item}(Y);$ $\text{sum} := \text{sum} + Y;$

← T_3 reads X after N is subtracted and reads Y before N is added; a wrong summary is the result (off by N).

Historias

Definición: Una historia (schedule) H , para un conjunto de ejecuciones de transacciones, es un orden parcial de las operaciones de las transacciones y que muestra cómo las transacciones son intercaladas. Si dos operaciones o_1 y o_2 de una transacción T ocurrían o_1 antes que o_2 en T entonces necesariamente ocurrirá o_1 antes que o_2 en H .

Operaciones Conflictivas: Dos operaciones son conflictivas si operan sobre el mismo ítem y al menos una de ellas es una escritura.

Equivalencia: Dos historias H_i y H_j son conflicto equivalentes $H_i \equiv H_j$ si:

- Están definidas sobre el mismo conjunto de transacciones.
- El orden de las operaciones conflictivas de transacciones no abortadas es el mismo.

“Lee de”: Dadas dos transacciones T_i y T_j decimos que T_i lee X de T_j si T_i lee X y T_j fue la última transacción que escribió X y no abortó antes de que T_i lo leyera.

Tipos de Historias

- **Historia serial:** Una historia es serial si las transacciones se ejecutan secuencialmente, sin entrelazar sus operaciones.
- **Historia serializable:** Una historia es serializable si es conflicto equivalente a una historia serial.
- Niveles de recuperabilidad:
 - **Recuperable (RC):** Una historia H es recuperable si cada transacción hace su commit después de que hayan hecho commit todas las transacciones de las cuales lee.
 - **Avoids Cascading Aborts (ACA):** Una historia H evita aborts en cascada (ACA) si toda transacción lee únicamente de transacciones que ya hayan commiteado.
 - **Stricta (ST):** Una historia H es estricta (ST) si toda transacción lee o escribe ítems únicamente luego de que la transacción que lo escribió previamente haya hecho commit o abort.
 - **Teorema:** $\text{Seriales} \subset ST \subset ACA \subset RC$

Verificación de Serializabilidad

Se utiliza el grafo de precedencia $SG(H)$. Es un grafo dirigido con las siguientes características:

- Un nodo para cada transacción $T_i \subseteq H$

- Hay ejes entre T_i y T_j sí y sólo sí hay una operación de T_i que precede en H a una operación de T_j y son operaciones conflictivas.
- Etiquetamos los ejes del grafo con los nombres de los ítems que los generan.

Teorema: Una historia H es **SR** sí y solo sí $SG(H)$ es acíclico.

Paradigma Pesimista

Definición: En el paradigma pesimista, se bloquea el registro para su uso exclusivo hasta que haya terminado con él. Este paradigma puede generar *deadlocks*.

Lock: Un lock es una variable asociada con un ítem de datos que describe el estado de ese ítem con respecto a posibles operaciones que puedan aplicarse a él.

Tipos de Locks

Lock Binario

Un lock binario es aquél que tiene dos posibles estados: *locked* & *unlocked*.

Una transacción T puede leer o escribir un ítem X si previamente realizó un lock sobre X y no lo ha liberado. Solo una T puede tener el lock sobre X al mismo tiempo.

Lock Ternario / Shared Lock

Un lock ternario tiene tres posibles estados:

1. *Read Locked* (o bloqueo compartido)
2. *Write Locked* (o bloque exclusivo)
3. *Unlocked*

Una transacción T puede leer un ítem X si hizo un *read lock* o *write lock* sobre X y no lo ha liberado, y además ninguna otra transacción tiene un *write lock* sobre X al mismo tiempo.

Una transacción T puede escribir un ítem X si hizo un *write lock* sobre X y no lo ha liberado, y además ninguna otra transacción tiene un *read lock* o *write lock* sobre X al mismo tiempo.

Two Phase Locking (2PL)

Definición: Una transacción respeta el protocolo de 2PL si todas las operaciones de bloqueo (lock) preceden a la primer operación de desbloqueo (unlock) en la transacción. Una transacción que cumple con el protocolo se dice que es una transacción 2PL

Serializabilidad: Dado T_1, T_2, \dots, T_n , si toda T_i es 2PL, entonces todo H legal sobre T_1, \dots, T_n es SR.

Variantes de 2PL

2PL Estricto (2PLE o S2PL)

Una transacción cumple con 2PLE si es 2PL y no libera ninguno de sus *write locks* hasta después de realizar el commit o el abort. 2PLE garantiza que la historia es **ST**.

2PL Riguroso (2PLR)

Una transacción cumple con 2PL Riguroso si es 2PL y no libera ninguno de sus *read & write locks* hasta después de realizar el commit o el abort.

Paradigma Optimista

Definición: En el paradigma optimista, se asume que no ocurrirá un comportamiento no serializable y sea actúa para reparar el problema sólo cuando ocurre una violación aparente.

Método: Timestamping

Cada transacción T tiene un único número llamado timestamp: $TS(T)$. Esta marca de tiempo es asignada en orden ascendente. Es decir si T_1 ocurre antes que T_2 , entonces $TS(T_1) < TS(T_2)$.

Cada elemento de la base de datos, X , debe asociarse a dos timestamp y un bit extra.

- **RT(X)**: tiempo de lectura, el timestamp más alto de una transacción que ha leído X .
- **WT(X)**: tiempo de escritura, el timestamp más alto de una transacción que ha escrito X .
- **C(X)**: bit de commit para X , es verdadero si y sólo si la transacción más reciente que escribió X ha realizado commit.

El planificador asume que el orden de llegada de las transacciones es el orden serial en que deberían parecer que se ejecutan. Ante la solicitud de una transacción T para una lectura o escritura, el planificador puede:

1. Conceder la solicitud
2. Abortar y reiniciar T con un nuevo timestamp (rollback)
3. Demorar T y decidir luego si abortar o conceder la solicitud (si el requerimiento es una lectura que podría ser sucia).

Reglas para el planificador

El planificador recibe una solicitud de lectura $rt(X)$

Caso 1: Si $TS(T) \geq WT(X)$ - es físicamente realizable es decir, no sucede read too late.

- Si $C(X)$ es *true*, conceder la solicitud. Si $TS(T) > RT(X)$ hacer $RT(X) = TS(T)$, de otro modo no cambiar $RT(X)$.
- Si $C(X)$ es *false* demorar T hasta que $C(X)$ sea verdadero o la transacción que escribió a X aborte.

Caso 2: Si $TS(T) < WT(X)$ - es físicamente irrealizable (read too late)

- Se hace Rollback T (abortar y reiniciar con un nuevo timestamp).

El planificador recibe una solicitud de escritura $wt(X)$

Caso 1: Si $TS(T) \geq RT(X)$ y $TS(T) \geq WT(X)$ - es físicamente realizable, no sucede write too late.

1. Escribir el nuevo valor para X .
2. $WT(X) := TS(T)$, o sea asignar nuevo WT a X .
3. $C(X) := false$, o sea poner en falso el bit de commit.

Caso 2: Si $TS(T) < RT(X)$ - es físicamente irrealizable, es decir, write too late.

- Se hace Rollback T (abortar y reiniciar con un nuevo timestamp).

Caso 3: Si $TS(T) \geq RT(X)$ pero $TS(T) < WT(X)$ - es físicamente realizable, pero ya hay un valor posterior en X .

- Si $C(X)$ es *true*, ignora la escritura.
- Si $C(X)$ es *false* demorar T hasta que $C(X)$ sea verdadero o la transacción que escribió a X aborte

El planificador recibe una solicitud de commit $C(T)$

Para cada uno de los elementos X escritos por T se hace:

- $C(X) := true$.

Se permite proseguir a las transacciones que esperan a que X sea committed

El planificador recibe una solicitud de abort o rollback $A(T)$ o $R(T)$

Cada transacción que estaba esperando por un elemento X que T escribió debe repetir el intento de lectura o escritura y verificar si ahora el intento es legal.

Método: Timestamping Multiversión

Variación del mecanismo de timestamp que mantiene versiones antiguas de los elementos de la base de datos. Permite leer la versión de X apropiada según el timestamp de T y de esta forma evita el read too late.

Modificación de las reglas para el planificador

- Cuando ocurre $wt(X)$, si es legal (según las reglas vistas) entonces se crea una nueva versión del elemento X . Su tiempo de escritura es $TS(T)$ y nos referimos a él como X_t , donde $t = TS(T)$.
- Cuando ocurre una lectura $rt(X)$ el planificador busca una versión X_t de X tal que $t \leq TS(T)$ y que no haya otra versión $X_{t'}$ tal que $t < t' \leq TS(T)$.
- Los tiempos de escritura están asociados a versiones de un elemento y nunca cambian.
- Los tiempos de lectura también son asociados con versiones. Lo podemos notar cómo $X_{t,tr}$ (donde tr es el último tiempo de lectura de X_t). Una transacción T' que quiere escribir debe hacer rollback cuando existe alguna $X_{t,tr}$ tal que $t < TS(T')$ y $tr > TS(T')$.

Borrado de versiones: Si una versión X_t tiene un tiempo de escritura tal que no existe una transacción activa T tal que $TS(T)$ sea menor t , se puede borrar cualquier versión de X previa a X_t .

Paradigma Optimista vs Paradigma Pesimista

Similitudes

- Ambas pueden generar *starvation* de una transacción.
- Ambas poseen un registro global que permite el control de concurrencia.
 - Pesimista: registro de locks.
 - Optimista:
 - Timestamping: timestamps de read/write y bit de commit.
 - Validación: read/write sets.

Diferencias

Generalmente el timestamping es mejor cuando la mayoría de las transacciones son de lectura o es raro que haya transacciones que traten de leer y escribir el mismo elemento.

En situaciones de mucho conflicto, locking suele comportarse mejor.

Se establece entonces un compromiso utilizado en los sistemas comerciales (transacciones read-only vs. transacciones read-write).

1. Transacciones read-write se manejan con locking pero crean versiones de los elementos.
2. Transacciones read-only se manejan con versiones creadas por transacciones read-write

Recuperación

Siempre que una transacción es recibida por el DBMS para su ejecución el sistema es responsable de:

- Que todas las operaciones de la transacción sean ejecutadas exitosamente y sus resultados sean almacenados en la BD (committed).
- ó, en caso contrario, que ninguna operación tenga efecto sobre la BD (aborted).

Ante un fallo en la transacción T previo a que se transforme en committed, el DBMS deberá deshacer los cambios realizados por las operaciones ejecutadas hasta el momento de T , dejando sin efecto lo hecho hasta ese momento y llevando la base de datos a un estado consistente.

Log del Sistema

El Log mantiene registro de todas las operaciones de las transacciones que afectan los valores de la BD. Permite restablecer el sistema ante fallos que afectan a las transacciones.

Características:

- El Log es un archivo append-only al que se le van agregando registros que dan cuenta de eventos importantes.
- Los posibles tipos de registros son:
 - <START T > La transacción T ha empezado.
 - <COMMIT T > La transacción T ha completado exitosamente.
 - <ABORT T > La transacción T abortó.
 - Update record: Indica que hubo un cambio en un ítem de la BD.
- Las transacciones se ejecutan concurrentemente, grabando los Logs Records en forma intercalada en el Log.

Políticas de Recuperación

Transacciones completas: son las transacciones commiteadas (terminaron exitosamente con COMMIT) y las transacciones abortadas (terminaron con ABORT).

Transacciones incompletas: son las transacciones con registro de START, y sin registro COMMIT ni ABORT.

Sin Checkpoint

	Undo	Redo	Undo/Redo
Descripción	Deshace las transacciones incompletas.	Rehace las transacciones que hicieron <i>commit</i> .	Deshace las transacciones incompletas y rehace todas las que fueron commiteadas
Registros en Log	$\langle T, X, ValorAnterior \rangle$	$\langle T, X, ValorNuevo \rangle$	$\langle T, X, ValorAnterior, ValorNuevo \rangle$
Reglas	1. Los registros del tipo $\langle T, X, v \rangle$ se escriben a disco antes que el nuevo valor de X se escriba a disco. 2. El registro $\langle Commit\ T \rangle$ se escribe a disco después que todos los elementos modificados por T se hayan escrito en disco (y tan	1. Los registros del tipo $\langle T, X, v \rangle$ se escriben a disco antes que el nuevo valor de X se escriba a disco. 2. El registro $\langle Commit\ T \rangle$ se escribe a disco antes que cualquier elemento modificado por T se haya escrito en disco.	Los registros del tipo $\langle T, X, v, w \rangle$ se escriben a disco antes que el nuevo valor de X se escriba a disco en la base de datos.

	rápido como sea posible).		
Recuperación	<ol style="list-style-type: none"> 1. Recorrer el log de abajo hacia arriba, recordando las transacciones completas. 2. Para cada $\langle T, X, v \rangle$ encontrado: <ul style="list-style-type: none"> ● Si T es completa, no hacer nada. ● Si no, cambiar el valor de X a v en la BD. 3. Escribir un registro $\langle \text{ABORT } T \rangle$ por cada T incompleta que no fue previamente abortada. 4. Flushear el log. 	<ol style="list-style-type: none"> 1. Identificar las transacciones commiteadas. 2. Recorrer el Log desde el comienzo y para cada registro update $\langle T, X, v \rangle$ encontrado: <ul style="list-style-type: none"> ● Si T es una transacción no commiteada, no hacer nada. ● Si T es una transacción commiteada, escribir v para el ítem X en la BD. 3. Para cada transacción incompleta T, escribir un registro $\langle \text{ABORT } T \rangle$ en el log y flushearlo. 	<ol style="list-style-type: none"> 1. Aplicar Redo a todas las transacciones commiteadas en el orden de las primeras a las últimas. 2. Aplicar Undo a todas las transacciones incompletas en el orden de las últimas a las primeras.
Ventajas	<ol style="list-style-type: none"> 1. En general, hay pocas transacciones incompletas. Tendremos que deshacer pocos cambios. 2. Podemos flushear datos a la BD durante la escritura del log. No hace falta demorar la escritura en BD hasta el final. 	?	Provee mayor flexibilidad para ordenar las acciones.
Desventajas	<ol style="list-style-type: none"> 1. Para recuperar, hay que leer todo el log. 2. Se requiere que los ítems sean grabados inmediatamente después de que la transacción terminó, quizás incrementando el número de I/O. 	<ol style="list-style-type: none"> 1. Para recuperar, hay que leer todo el log (2 veces). 2. En general, hay muchas transacciones completas. Tendremos que rehacer muchos cambios. 3. Debemos demorar la escritura en la BD, hasta luego de escribir el COMMIT en disco. Esto implica mantener todos los bloques de la BD modificados en el buffer, hasta el COMMIT. 	<ol style="list-style-type: none"> 1. Se guarda más información en el Log. 2. Más trabajo en caso de un crash.

Undo / Checkpoint Quiescente

Checkpoint Quiescente: No aceptan nuevas transacciones (el sistema queda "inactivo" durante el checkpoint).

Etapas:

1. Dejar de aceptar nuevas transacciones.
2. Esperar a que todas las transacciones activas commiteen o aborten.
3. Escribir un $\langle \text{CKPT} \rangle$ en el log y luego efectuar un flush.
4. Aceptar nuevas transacciones.

Recovery: Se lee el Log a partir del último registro y hasta el punto de checkpoint. Sabemos que hasta ahí todas las transacciones terminaron y sus cambios fueron guardados en disco. Aplicar la política del UNDO.

Checkpoint No Quiescente

	Undo	Redo	Undo/Redo
Etapas	1. Escribir <Start CKPT(T1,T2,...,Tk)> en el log, y efectuar un flush. T1,T2,...,Tk son las transacciones activas (sin commit ni abort) en el momento.		
	2. Esperar a que todas las transacciones T1,T2,...,Tk terminen (ya sea abortando o commiteando).	2. Esperar a que todas las modificaciones realizadas en los buffers por transacciones ya commiteadas al momento del Start CKPT sean escritas a disco.	2. Escribir a disco todos los buffers "sucios" (contienen elementos cambiados que aún no fueron escritos a disco) al momento del START CKPT. Escribir incluso datos modificados por transacciones no commiteadas.
	3. Escribir <End CKPT> en el log y efectuar un flush.		
Semántica del START CKPT	Las transacciones terminadas están en disco. Las activas con T1, ..., Tk	Puede haber transacciones terminadas no en disco. Las activas son T1, ..., Tk	Puede haber transacciones terminadas no en disco. Las activas son T1, ..., Tk
Semántica del END CKPT	Las transacciones T1, ..., Tk están en disco (se pudieron haber iniciado otras entre START y END).	Todas las terminadas que no estaban en disco al momento del START ahora sí lo están. No sabemos nada sobre T1, ..., Tk.	Todas las terminadas que no estaban en disco al momento del START ahora sí lo están. No tenemos certezas sobre T1, ..., Tk.
Recovery	UNDO, leyendo el log desde el último registro. ● Encuentro un <END CKPT>, debo leer no más allá del <START CKPT>. ● Encuentro un <START CKPT> (no hay <END CKPT> por crash). Debo leer el Log hasta el <START T _i > más antiguo de las transacciones de la lista del <START CKPT> que quedaron incompletas.	Leer el log de abajo hacia arriba, buscando un <END CKPT>. ● Si lo encuentro, leemos hasta el <START CKPT(T1, ..., Tk)> asociado. Luego, hacemos una recuperación Redo tradicional, desde el <START Ti>, i = 1, ..., k, más lejano tal que Ti haya hecho commit. ● Si encontramos un START CKPT sin un END CKPT, lo ignoramos y seguimos buscando un END CKPT..	Se aplica la política UNDO/REDO. 1. Transacciones incompletas: para deshacerlas deberemos retroceder hasta el start más antiguo de ellas. Agregar registro ABORT al log para cada transacción incompleta, y hacer flush del log. 2. Transacciones commiteadas: Si leyendo desde el último registro del log, encontramos un registro <End CKPT> sólo será necesario rehacer las acciones efectuadas desde el correspondiente registro Start CKPT en adelante.

7. Optimización

El Query Optimizer

Una query suele tener muchas posibles formas de ser ejecutada para obtener el set de datos deseado. Cada una de esas formas de ejecución se llama query plan. Uno de los componentes del DBMS, el query optimizer es el encargado de seleccionar cual de esos planes ejecutar. Cómo encontrar el plan óptimo es un problema NP-Completo, el query optimizer utiliza otras técnicas para encontrar uno suficientemente bueno en un tiempo razonable. La optimización de la query tiene varios aspectos:

- **Utilización de heurísticas:** se aplican transformaciones del álgebra relacional que tienen la propiedad de mantener los resultados obtenidos. Las heurísticas suelen mejorar la performance, pero no es una garantía.
- **Estimación de selectividad:** el optimizer utiliza la información almacenada en el catálogo de la base de datos para estimar el grado de selectividad que tiene la query ("cuantas tuplas devuelve"). De esta forma se puede tener una medida estimativa de cuán "cara" va a ser la ejecución de dicha query.
- **Índices y tipo de archivo:** el plan de ejecución elegido depende muy fuertemente de los índices que se disponga en la tabla y cómo estén ordenados físicamente los datos en disco.

Una vez obtenido el plan de ejecución, el code generator genera el código correspondiente a ese plan y el runtime database processor lo ejecuta.

Relaciones

Parámetros para medir (R es una relación):

- Bloque: porción de datos que se levanta en cada lectura de disco.
- LB: longitud de bloque.
- B_R : cantidad de bloques que ocupa R.
- FB_R : factor de bloqueo de R. Es la cantidad de tuplas de R que entran en un bloque.
- L_R : longitud de una tupla de R.
- T_R : cantidad total de tuplas de R.
- $I_{R,A}$: imagen del atributo A de R. Es la cantidad de valores distintos de la columna A en R.
- X_I : altura del árbol de búsqueda I (I un índice árbol B+)
- FB_I : factor de bloqueo de I. Es la cantidad de tuplas de I que entran en un bloque.
- BH_I : cantidad de bloques que ocupan todas las hojas de I (I un índice árbol B+).
- $MBxB_I$: cantidad máxima de bloques en un bucket de I (I un índice hash).
- CBu_I : cantidad de buckets de I (I un índice hash).
- B: cantidad de bloques que entran en memoria principal.

Almacenamiento físico (Archivos)

rid: Cada tupla en una relación tiene asociado un identificador, llamado *rid* (register id). Este *rid* no es un atributo de la relación.

	Heap File	Sorted File
Descripción	Los archivos Heap son el tipo de archivo más simple, que consiste de una colección desordenada de registros, agrupados en bloques.	Los archivos Sorted contienen los registros ordenados de acuerdo a los valores de determinados campos.
Costo de exploración completa	B_r	B_r
Costo de búsqueda por igualdad ($A = c$)	B_r	$\log_2(B_r) + \lceil T' / FB_r \rceil$
Costo de búsqueda por rango ($c \leq A \leq d$)	B_r	$\log_2(B_r) + \lceil T' / FB_r \rceil$

- T' es la cantidad de tuplas que cumplen con el criterio de búsqueda
- FB_r es la cantidad de tuplas por bloque de R

Índices

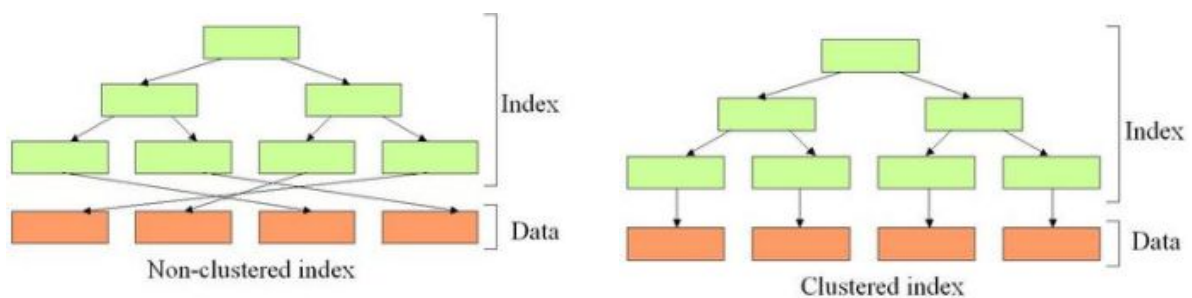
Es un diccionario (con claves no necesariamente únicas) asociado a una relación.

- La clave es una o más columnas de la relación.
- El valor asociado a una clave es el registro (la tupla) asociado a esa clave. El valor puede tener varias formas:
 - El registro completo.
 - El rid del registro.
 - Una lista de los rid de todos los registros con esa clave.

Propiedades

Índices Clustered vs Unclustered

Si los datos del archivo están ordenados físicamente en el mismo orden que uno de sus índices, decimos que ese índice es clustered. Caso contrario es unclustered. Los archivos de datos a lo sumo pueden tener un índice clustered, en tanto que la cantidad de índices unclustered es ilimitada



Índices Densos vs No Densos

Un índice es denso si tiene una entrada por cada registro en la tabla de datos. Por otro lado, un índice no denso tiene entradas solo para algunos registros.

Índices Primarios vs Secundarios

Índice primario: los valores que almacena son los registros completos de los archivos. Solo puede haber uno por tabla.

Índice secundario: los valores son rids. Puede haber más de uno por tabla.

Representaciones

Índice Árbol B+

Definición: Árbol de búsqueda balanceado, con una cantidad de hijos y claves por nodo dada por un parámetro d (cada nodo interno, excepto la raíz, tiene entre $d/2$ y d claves e hijos).

- Los nodos hoja contienen la información asociada a las claves de todos los registros del archivo.
- Las hojas se pueden recorrer como si formaran una lista doblemente enlazada.
- Estos índices son los recomendados para acceder a rangos de claves.

	Clustered	Unclustered
Exploración completa	El índice no nos sirve. Utilizar file scan.	
Búsqueda por igualdad ($A = c$)	$X_i + \lceil T' / FB_r \rceil$	$X_i - 1 + \lceil T' / FB_l \rceil + T'$
Explicación	Bajar en el árbol, buscando la primer ocurrencia de la clave. Ir a buscar el registro asociado en el archivo. Recorrer el archivo a partir de ese punto, mientras encuentre ocurrencias de la clave.	Bajar en el árbol, buscando la primer ocurrencia de la clave. A partir de esa clave en esa hoja, recorrer secuencialmente las hojas, mientras encuentre ocurrencias de la clave. Por cada clave encontrada, leer el registro asociado en el archivo.
Búsqueda por rango	$X_i + \lceil T' / FB_r \rceil$	$X_i - 1 + \lceil T' / FB_l \rceil + T'$
Explicación	Buscar el primer elemento del rango y luego recorrer el archivo a partir de ahí.	Buscar el primer elemento del rango, y a partir de ahí todas las ocurrencias. Por cada una, leer el registro asociado en el archivo.

Índice basado en hash estático

Definición: Tabla de hash, formada por una cantidad estática de buckets.

- Ideales para búsqueda por igualdad, i. e. encontrar todas las claves k tal que $k = a$.

Cuadro resumen

Tipo de archivo / índice	Costo de exploración completa	Costo de búsqueda por igualdad ($A = k$)	Costo de búsqueda por rango ($k_1 \leq A \leq k_2$)
Heap file	B_R	B_R	B_R
Sorted file	B_R	$\log_2(B_R) + \lceil T' / FB_R \rceil$	$\log_2(B_R) + \lceil T' / FB_R \rceil$
Índice B+ clustered sobre A	-	$X_I + \lceil T' / FB_R \rceil$	$X_I + \lceil T' / FB_R \rceil$
Índice B+ unclustered sobre A	-	$X - 1 + \lceil T' / FB_I \rceil + T'$	$X - 1 + \lceil T' / FB_I \rceil + T'$
Índice hash estático sobre A	-	$MB \times B_I + T'$	-

Donde:

- T' es la cantidad de tuplas que cumplen con el criterio de la búsqueda
- $FB_R I$ es el factor de bloqueo del archivo
- FB_I es el factor de bloqueo del índice I
- $MB \times B_I$ es la cantidad máxima de bloques de un bucket

File Scan e Index Scan

- **File scan:** Recorrido sobre las entradas de un archivo.
- **Index scan:** Recorrido sobre las entradas de un índice.

Index matching:

- Tenemos una selección y queremos usar un índice.
- Podemos hacerlo en dos casos:
 - Tenemos un índice árbol B+, y el predicado es una conjunción de términos (atrib op valor) (op es =, < ó >) que involucra a atributos de un prefijo de la clave del índice (o sea, si la clave es una tupla, involucra a todos los atributos que forman un prefijo de la tupla).
 - Tenemos un índice hash, y el predicado es una conjunción de términos que involucra a todos los atributos del índice.

Join

Block Nested Loop Join(BNLJ)

Es el approach de fuerza bruta. Si se tienen B bloques de memoria, se llenan $B - 2$ con bloques de una de las tablas. Otro bloque se usa para ir iterando todos los bloques de la otra tabla y el restante para el resultado. Siempre conviene poner el archivo con menos bloques en la iteración exterior (o sea llenar los $B - 2$ con él).

Index Nested Loop Join(INLJ)

Se utiliza cuando se tiene un índice en una de las tablas que coincida con el atributo del join. Se itera sobre la otra tabla, buscando los atributos coincidentes utilizando el índice. El costo depende del tipo de índice.

Sort Merge Join(SMJ)

Se ordenan ambas relaciones (si no estaban ordenadas) y se recorren ordenadamente. El costo es el de ordenar ambas relaciones (algoritmo de sorting en disco) y luego hacer el merge (lineal en ambos tamaños).

Pipeline & Materialize

Materialización: Es escribir el resultado de una operación en disco.

Pipelining: Si entre dos operaciones tenemos un pipeline, la primer operación O1 va pasándole las tuplas resultado en forma de stream, a la segunda operación O2. Este pasaje se hace completamente en memoria, lo cual evita escribir el resultado de la primer operación en disco. Por ende, el costo de output de O1 es 0, al igual que el costo de input de O2.

- No toda operación se puede pipelinizar. Hay operaciones que necesitan tener todas las tuplas de una relación de antemano, para poder empezar a realizar el cómputo. Por ejemplo, los joins (siempre hay que materializar el input de los joins).

Optimizaciones Algebraicas

Buscan mejorar la performance de la consulta independientemente de la representación física. Se basan en propiedades algebraicas que transforman la consulta en AR en una equivalente.

- Cascada de σ
 - $\sigma_{C1 \text{ and } C2}(R) = \sigma_{C1}(\sigma_{C2}(R))$
- Conmutatividad de σ
 - $\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C2}(\sigma_{C1}(R))$
- Cascada de π
 - $\pi_{L1 \cap L2}(R) = \pi_{L1}(\pi_{L2}(R))$
- Conmutatividad de σ con respecto a π
 - $\sigma_C(\pi_{A1, \dots, An}(R)) = \pi_{A1, \dots, An}(\sigma_C(R))$
siempre que C sólo use atributos de A1, ..., An.
- Conmutatividad del producto cartesiano (o junta)
 - $R \times S = S \times R$
- Conmutatividad de la unión e intersección
 - $R \text{ op } S = S \text{ op } R$
donde op es \cup o \cap .
- Asociatividad de producto cartesiano, junta, unión e intersección
 - $(R \text{ op } S) \text{ op } T = R \text{ op } (S \text{ op } T)$
donde op es \times , join, \cup o \cap .
- Conmutatividad de σ con respecto al producto cartesiano (o junta)
 - $\sigma_C(R \times S) = \sigma_{C1}(R) \times \sigma_{C2}(S)$
donde $C1 \cup C2 = C$, y cada Ci son las condiciones que tienen sentido en cada relación.
- Conmutatividad de π con respecto al producto cartesiano (o junta)
 - $\pi_L(R \times S) = \pi_{L1}(R) \times \pi_{L2}(S)$
donde $L1 \cup L2 = L$, y cada Li son los atributos que tienen sentido en cada relación.

Heurísticas

- Considerar sólo árboles sesgados a izquierda. Ésto limita los árboles candidatos a analizar.
- Descomponer las selecciones conjuntivas. Esto puede ayudar, por ejemplo, si tenemos que seleccionar con una condición sobre dos atributos A y B, pero sólo tenemos un índice sobre A, podemos separar las selecciones y usar el índice.
- Llevar las selecciones lo más cercano posible a las hojas del árbol. Ésto ayuda a reducir la cantidad de datos que hay que materializar hacia arriba, seleccionando prematuramente las tuplas que nos interesan.
- Reemplazar productos cartesianos seguidos de selecciones por joins.
- Descomponer las listas de atributos de proyecciones y llevarlas lo más cerca de las hojas posibles.
- Realizar primero los joins más selectivos.
- Usar pipeline entre operaciones siempre que sea posible.

8. NoSQL

NoSQL: Término utilizado para referirse a BD que no siguen los principios de los RDBMS

- Sistemas diseñados para lograr alta velocidad y escalabilidad.
- Usualmente NO respetan propiedades ACID de transacciones

Propiedades BASE

- **Basic Availability:** Cada solicitud garantiza una respuesta: ejecución exitosa o fallida.
- **Soft-state:** El estado del sistema puede cambiar con el tiempo, a veces sin ninguna entrada (por consistencia eventual).
- **Eventual consistency:** La BD puede ser momentáneamente inconsistente pero será consistente con el tiempo.

Modelo Relacional vs NoSQL

Problemas de RBDS:

- Cumplir con las propiedades ACID es costoso. El cumplimiento actúa el detrimento:
 - La performance del sistema.
 - La disponibilidad del sistema.
- Las queries suelen involucrar joins, que también son muy costosas.

Solución: BASE

Algunas ventajas de NoSQL:

- **Representación de datos libre de esquemas:** La estructura puede ir evolucionando fácilmente en el tiempo, e incluso pueden convivir datos con distintas estructuras.
- **Tiempo de desarrollo:** Al ser la representación de datos más adecuada para el problema, se pueden evitar ciertos JOINS extremadamente complejos.
- **Velocidad:** En muchos casos es posible dar respuesta en el orden de los milisegundos en vez de cientos de milisegundos.
- **Planificar escalabilidad:** La aplicación puede ser bastante elástica y manejar picos repentinos de carga.

Taxonomías

Clave - Valor

Funcionamiento: Se vincula una clave con un valor (similar a un diccionario).

Operaciones principales:

- *put(key, value)*
- *get(key)*
- *delete(key)*

Características:

- Sólo se permite operar a través de la clave
- Se puede almacenar cualquier tipo de datos en el campo valor
- Escala: No es necesario revisar todas las entradas para consultar un valor

Column Store

Funcionamiento: Almacenan la información en columnas (a diferencia de las BD orientadas a filas que almacenan los datos en forma de fila).

Ventajas: Consultas de tipo agregación (máximo, mínimo, promedio, etc) tienen muy buena performance.

Column Family Store

Funcionamiento:

Las Column Family Store son especializaciones de las Column Store que se comportan de forma similar a key-value, pero usando los siguientes campos como clave para los datos.

- Nombre / id de fila
- Nombre / id de columna
- Column family: permite agrupar varias columnas y de esa forma poder realizar queries que afecten a todas
- Timestamp: permite almacenar distintas versiones del valor a través del tiempo

Características:

- No es necesario ingresar valores para todas las columnas (la implementación se puede pensar como una matriz esparsa).
- Escala muy bien cuando la matriz posee muchas celdas vacías (a diferencia de RDBMS).

Ventajas:

- No existe la necesidad de hacer joins, por lo que escala bien en sistemas distribuidos.
- Capacidad de replicar datos en múltiples nodos de la red. La ausencia de JOINS permite almacenar cualquier parte de la matriz en equipos remotos.

Documentos

Funcionamiento: Los datos se agrupan en colecciones de documentos. Un documento es un registro con datos semi-estructurados (por ejemplo XML o JSON).

Características:

- Los documentos no necesariamente tienen todos la misma estructura.
 - Esto se debe a que no soportan esquemas, y por lo tanto no poseen validación contra un esquema (potencialmente negativo).
- Se puede consultar por cualquier valor de los documentos.
 - Se suelen utilizar índices sobre los valores.

Gráfos

Funcionamiento: Un graph store consiste en muchas estructuras del tipo nodo-relación-nodo. Permiten almacenar relaciones y trabajar con ellas de manera muy eficiente. Los nodos suelen representar entidades, y las relaciones pueden ser consideradas como las conexiones entre estas entidades.

Ventajas: Resultan muy útiles cuando el núcleo de nuestro problema son las relaciones entre los objetos del dominio.

- Por ejemplo el modelado de redes sociales, problemas basados en reglas, etc.
- Si se usaran RDBMS, el costo de responder las misma consultas sería muy alto, dado el costo de los JOINS.

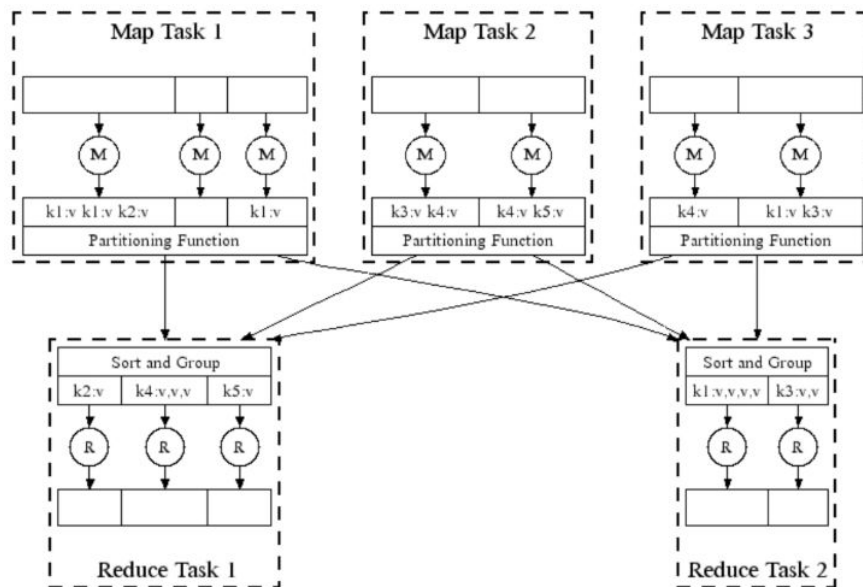
Desventajas: Son difíciles de hacer escalar, puesto que no es fácil particionar un grafo en varios servidores de modo tal de no afectar la performance general de las queries.

Herramientas / Operaciones

Map - Reduce

Definición: MapReduce es un marco de trabajo para el procesamiento en paralelo de grandes volúmenes de datos en varios equipos (nodos).

- Map recupera los datos de la BD y los transforma en una colección de operaciones que pueden ser ejecutados independientemente en distintos procesadores. La salida de los map, son pares (clave, valor).
- Como siguiente fase, Reduce, utiliza los pares como entrada, ejecuta la operación asignada y retorna un resultado.



Sharding

Definición: Particionado de la BD en fragmentos denominados *shards* que luego son distribuidos a través de servidores. Surge a partir de la imposibilidad de almacenar todos los datos en la misma máquina.

Características:

- Un shard es un fragmento de todos los datos. Los shards comparten esquema y su unión representa la totalidad del dataset.
- Debemos determinar a qué shard mandamos cada dato. Para esto se utiliza un hash de algún atributo de los datos. Por ejemplo la primera letra del nombre de usuario, o la ubicación geográfica.
 - ¿Qué pasa si el atributo cambia?

Ventajas:

- Permite escalar horizontalmente.
- Provee tolerancia parcial ante fallos, pues si un nodo falla, sólo su porción de los datos quedan fuera de servicio.

Desventajas:

- Consultas que involucran varios nodos pueden afectar negativamente la performance.

Réplicas

Definición: Almacenamiento de múltiples copias de la BD, cada una de ellas conocidas como *réplicas*. Surge a partir del hecho de que a medida que aumenta la cantidad de servidores, aumenta la probabilidad de falla.

Método de implementación: Master - Slave

Funcionamiento:

- Escritura (insert, delete, update) en el nodo *Master*.
- Lectura sobre cualquier nodo *Slave*.

Ventajas:

- Ideal para escenarios de uso intensivo de lecturas.
- Si el nodo *Master* falla, se puede continuar con las lecturas.
- Nodo *Slave* configurado como backup puede tomar el rol de *Master* en caso de fallo.

Desventajas:

- El nodo *Master* puede resultar un cuello de botella frente a una gran cantidad de escrituras.
- Las lecturas pueden ser inconsistentes, debido a la diferencia de tiempo para recibir las actualizaciones en cada réplica.
 - Posible solución: un sistema de votación donde una lectura es consistente si la mayoría de los nodos contienen la misma versión del registro.
 - Contra: requiere sistema de comunicación entre nodos *Slave* rápido y confiable

Método de implementación: P2P (Peer To Peer)

Funcionamiento: Todos los nodos (denominados *peers*) poseen el mismo nivel de jerarquía y son capaces de manejar tanto lecturas como escrituras.

Ventajas:

- Al ser descentralizado, no hay cuellos de botella. Mayor tolerancia a fallas.
- Mayor throughput, porque todos los nodos pueden responder peticiones.

Desventajas:

- Lecturas inconsistentes y escrituras conflictivas.
 - Solución 1: Estrategia de concurrencia pesimista.
 - Utilización de locks..
 - Va en contra de la disponibilidad.
 - Solución 2: Estrategia de concurrencia optimista.
 - No utiliza locks
 - Permite inconsistencias sabiendo que eventualmente todas las actualizaciones de propagarán y se llegará a un estado consistente
 - Para asegurar consistencia, se puede implementar un sistema de votación como en *Master - Slave*.

Sharding + Réplicas

Funcionamiento: Por cada shard creamos varias réplicas. Esto permite combinar la escalabilidad que provee el sharding, con la disponibilidad de la replicación.

9. Bases de Datos Distribuídas (DDB)

Definición: Colección de múltiples BD que están lógicamente relacionadas y se encuentran distribuídas en una red de computadoras.

Sistema de Gestión de DDB (DDBMS): Software que permite gestionar la DDB y que hace transparente la distribución al usuario

Condiciones que debe satisfacer una DDB:

- Interconexión entre BDs a través de una red de computadoras
- Relación lógica entre las BDs interconectadas
- Posible heterogeneidad entre los nodos

Beneficios de las DDBs:

- Aumento en la disponibilidad
 - Al aislar los fallos en sus sitios de origen, sin afectar las BDs de otros nodos.
- Mejora en la performance
 - Un DDBMS fragmenta la BD manteniendo los datos cerca de donde más se necesitan.
- Facilitación de escalabilidad vía expansión.
 - Expandir el sistema en términos de datos (como incrementar el tamaño del dataset) puede llegar a ser más sencillo que en un esquema centralizado.
- Transparencia: Ocultamiento de detalles de implementación a los usuarios finales
 - BD Centralizadas: Independencia entre datos físicos y lógicos
 - BD Distribuídas: Datos y Software están distribuidos, entonces se agregan nuevos tipos de transparencia
 - Organización de los datos
 - Fragmentación
 - Replicación
 - Otras
 - Transparencia total provee al usuario de una visión del DDBS como si fuera un sistema centralizado.
 - Existe un compromiso entre facilidad en el uso y el sobre costo de proveer transparencia

Fragmentación

Fragmentos: Partes que resultan de dividir la BD en unidades lógicas.

Tipos:

- Fragmentación Horizontal (o sharding): Los sitios poseen distintas tuplas / filas.
- Fragmentación Vertical: Los sitios poseen distintas columnas.
- Fragmentación Mixta o Híbrida: Combinación de las anteriores.

Replicación

La replicación de los datos permite a ciertos datos ser almacenados en más de un sitio, para incrementar la disponibilidad y confiabilidad.

Replicación total

Idea: La DDB se encuentra replicada en cada sitio.

Ventajas:

- Sistema continúa operando mientras haya al menos un sitio disponible.
- Mejora la performance de las lecturas (pueden realizarse de manera local).

Desventajas:

- Baja performance en las escrituras (tienen que realizarse en todas las copias).
- Las técnicas de Control de Concurrencia y Recovery son más costosas.

Replicación parcial

Idea: Algunos fragmentos son replicados, mientras que otros no.

Distribución de Datos: Proceso por el cual cada fragmento (o copia del fragmento) debe ser asignado a un sitio particular. La descripción de la replicación se guarda en el *Esquema de Replicación*.

Control de Concurrencia

Copia Distinguida

En los métodos de copia distinguida se designa una copia de cada ítem como copia distinguida. Los pedidos de *lock* y *unlock* son enviados al sitio de la copia distinguida.

	Sitio Primario	Sitio Primario + Backup	Copia Primaria
Descripción	Todas las copias distinguidas se mantienen en un único sitio.	Además de tener un sitio primario, se asigna un sitio como backup.	Las copias distinguidas pueden vivir en diferentes sitios.
Funcionamiento	<ul style="list-style-type: none">* Si T obtiene $read_lock(X)$ de sitio primario, puede acceder a cualquier copia de X.* Si T obtiene $write_lock(X)$ de sitio primario y actualiza a X el DDBMS es responsable de actualizar todas las copias antes de realizar $unlock(X)$.* Si todas las T siguen protocolo 2PC, la seriabilidad está garantizada.	Información sobre <i>locks</i> se mantiene en ambos sitios. En caso de falla el backup como el control.	Similar a Sitio Primario
Ventaja	Simple extensión del enfoque centralizado.	Evita que se paralice el sistema en caso de la falla del sitio primario.	Falla de un sitio solo afecta a los ítems con copia primaria en dicho sitio.
Desventaja	<ol style="list-style-type: none">1. Cuello de botella2. Una falla en el sitio primario	<ol style="list-style-type: none">1. Cuello de botella2. Disminuye	Una T puede necesitar <i>locks</i>

	paraliza el sistema.	performance en procesos de lock	en varios sitios.
--	----------------------	---------------------------------	-------------------

Votación

Funcionamiento:

- Se hace pedido de *lock(X)* a todos los sitios que contienen copia de X
- Cada copia de X es responsable de su propio lock y puede aceptar o denegar locks
- Si T pide un *lock(X)*:
 - Si recibe OK de mayoría de las copias, toma el lock y avisa a todas las copias que lo tomó
 - Si NO Recibe el ok de la mayoría de las copias dentro de un cierto tiempo, cancela su pedido e informa a todos los sitios de la cancelación

Ventaja:

- Es considerado un método de CC realmente distribuido (decisión reside en los nodos).

Desventajas:

- Mayor tráfico de mensajes.
- Tener en cuenta caída de sitios durante votación torna al algoritmo muy complejo.

Recovery

Problemas a resolver:

- Fallos en sitios y comunicaciones.
- Commit distribuido.
 - No se puede realizar un commit hasta asegurarse que los efectos de *T* no se van a perder en **ningún** sitio.

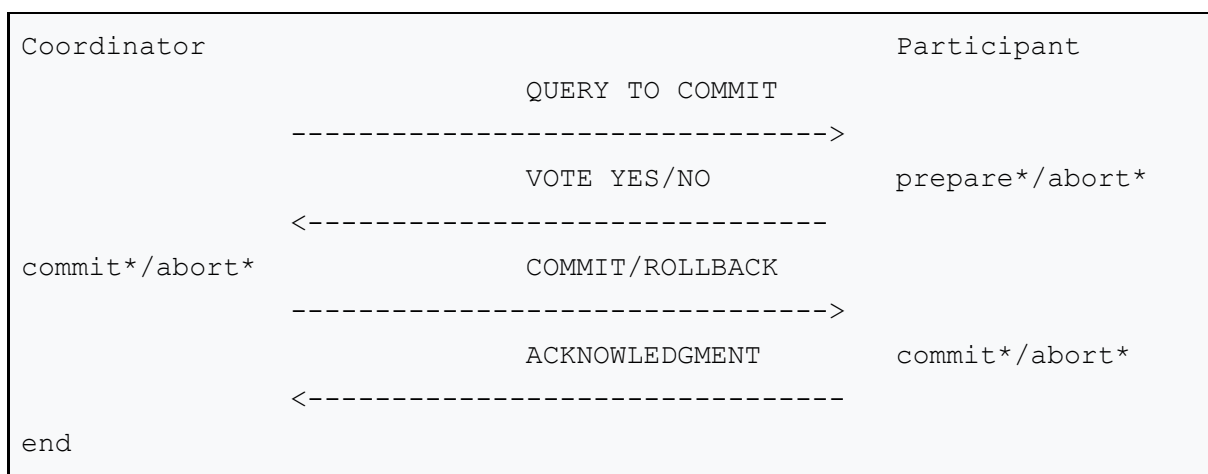
Solución: 2PC (two phase commit)

Transacciones

2PC

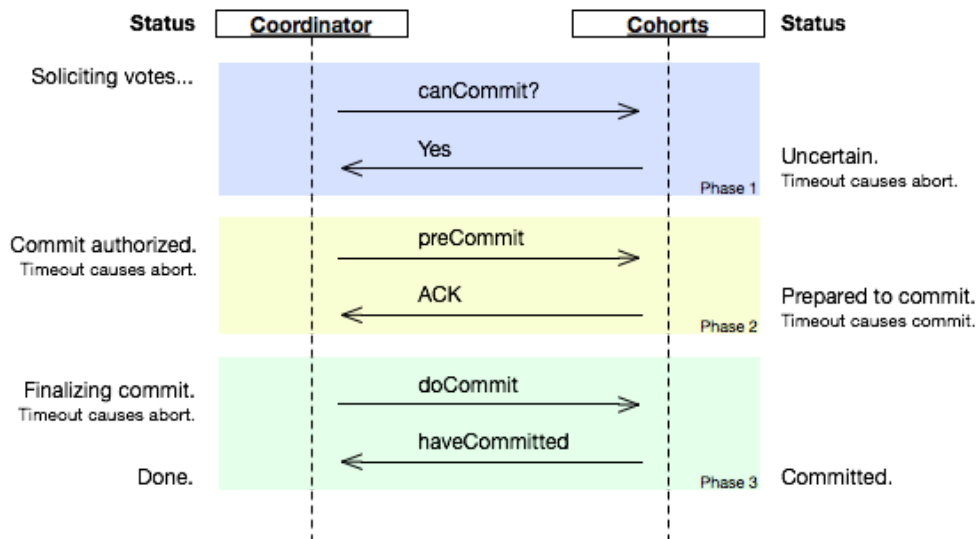
2PC es un protocolo mediante el cual el transaction manager se asegura de que todos los sitios puedan commitear una transacción antes de pedirles que efectivamente lo hagan.

Ejemplo:



Problema: Si falla el coordinador una vez pedido el lock, todos los sitios participantes quedan bloqueados. Solución: 3PC.

3PC



Procesamiento de Queries

1. **Mapeo:** Sea una consulta SQL de la DDB, se mapea a una consulta algebraica referenciando al esquema conceptual global. Es idéntico al paso en bases centralizadas
2. **Localización:** Se mapea el Query del paso anterior a múltiples queries sobre fragmentos individuales (se utiliza información de distribución y replicación de datos).
3. **Optimización de Query Global:** Selección de una estrategia de una lista de candidatos cercanos al óptimo. Tiempo es la unidad más utilizada para medir costo. Costo total es una ponderación de costos de CPU, I/O y comunicación.
 - a. Algoritmos de Optimización de Queries deben considerar reducir la “cantidad de datos a transferir”
4. **Optimización de Query Local:** Se realiza en todos los sitios de la DDB. Técnicas similares a las de BDs centralizadas

Tipos de DDB

Definición Común: Datos y Software distribuidos en múltiples lugares, pero conectados a través de una red de comunicaciones

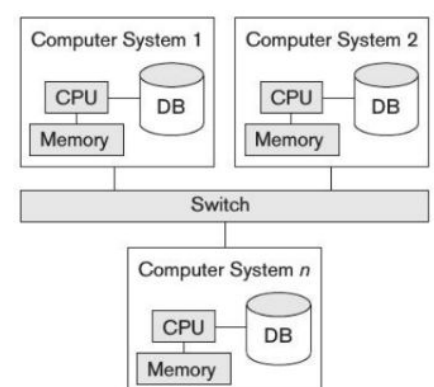
Factores que distinguen a estos sistemas entre sí:

1. **Grado de Homogeneidad:** Si todos los servers y usuarios utilizan el mismo software (Homogéneo vs. Heterogéneo).
2. **Grado de Autonomía Local:** Si no se le permite al sitio local funcionar como un DBMS standalone, entonces el sistema no posee autonomía local.

Paralelas vs Distribuidas

Bases de datos Paralelas:

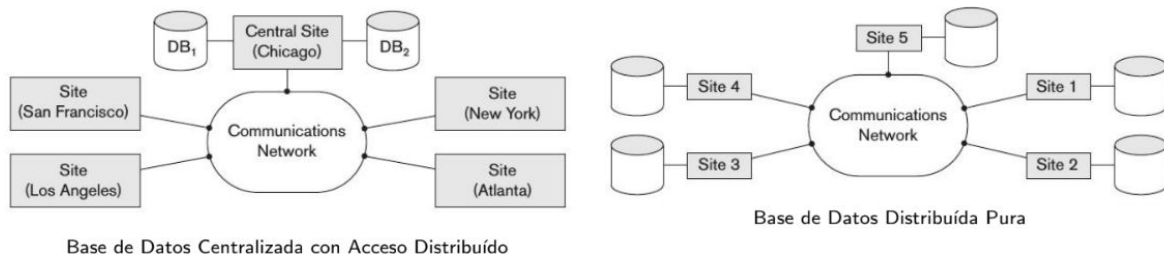
- Los servidores están físicamente cerca (por ejemplo, en la misma sala de servidores).
- Los servidores se conectan mediante una LAN dedicada de alta velocidad y switches



- Se asume que el costo de comunicación es bajo
- Pueden compartir memoria, disco, o nada (esta última se llama shared-nothing architecture).

Bases de datos distribuidas:

- Los servidores pueden estar lejos entre ellos. Por ejemplo, en diferentes continentes.
- Pueden estar conectados a través de internet.
- Los problemas y costos de comunicación no pueden ser ignorados.
- Casi siempre la arquitectura es shared-nothing.



Catálogo

Catálogo Global: Lugar donde se almacena la información acerca de la fragmentación, asignación de fragmentos y replicación de los datos.

Esquemas de Administración de Catálogos de DDBMS

- **Centralizado:** El esquema completo es almacenado en un único sitio.
 - Ventajas:
 - Simple de implementar
 - Desventajas:
 - Confiabilidad
 - Disponibilidad
 - Autonomía
 - Distribución de la carga de procesamiento
 - Locks pueden generar un cuello de botella en aplicaciones escritura-intensivas
- **Totalmente replicado:** En cada sitio se almacena una copia del catálogo completo
 - Ventajas:
 - Lecturas pueden responderse localmente
 - Desventajas:
 - Actualizaciones en el catálogo deben ser transmitidas a todos los sitios
 - Esquema centralizado de 2-Phase Commit para mantener consistencia del catálogo
 - Aplicaciones escritura-intensivas (locks) pueden causar un incremento en el tráfico de la red
- **Parcialmente replicado:** Cada sitio mantiene la información completa del catálogo de los datos que están almacenados de manera local en ese sitio
 - Cada sitio, permite también, almacenar en cache entradas obtenidas de otros sitios
 - El sistema lleva registro de las entradas del catálogo para los sitios donde se creó el objeto y para los sitios que contienen copias de este objeto

- Cualquier cambio en las copias se propaga inmediatamente al sitio original (de creación)

Teorema CAP

Un sistema distribuido no puede garantizar las siguientes propiedades simultáneamente:

- **Consistency (Consistencia):** Todos los nodos ven los mismos datos al mismo tiempo.
- **Availability (Disponibilidad):** Se garantiza que cada petición a un nodo recibe una confirmación de si ha sido o no resuelta satisfactoriamente.
- **Partition tolerance (Tolerancia a Partición):** El sistema continúa trabajando a pesar de un mensaje perdido o de una falla parcial.

10. Extensiones y Otras Bases

ORM (Mapeo Objeto - Relacional)

Definición: Las bases de datos objeto - relacional son un intento de obtener lo mejor de dos mundos:

- El modelo relacional, que soporta queries sofisticadas de alto nivel.
- El modelo orientado a objetos, que permite crear tipos de datos completos.

Características:

- Permite un modelado más intuitivo para aplicaciones con datos complejos.
- Retiene el fundamento matemático del modelo relacional.
- Permite atributos no atómicos o complejos: colecciones (arreglos, sets, etc.) y structs (tipos de datos definidos por el usuario).
 - De esta forma viola la 1FN (donde todos los argumentos son atómicos).
- Permite features de POO, como herencia.
- Permite el uso de atributos de tipo referencia. Estos atributos son una referencia verdadera a objetos en otras tablas.
 - Esto permite evitar hacer joins. En lugar de eso, usamos path expressions (atrib_1 -> atrib_2 para viajar del atrib_1 de una tabla, al objeto apuntado atrib_2 de otra tabla).

Ejemplo de structs:

```
create type Name as
  (firstname varchar(20),
   lastname varchar(20))
```

```
create table person (
  name Name,
  dateOfBirth date)
```

Ejemplo de herencia:

```
create type Person
  (name varchar(20),
   address varchar(20))
```

```
create type Student under Person
  (degree varchar(20),
   department varchar(20))
```

XML (Extensible Markup Language)

	Datos estructurados	Datos semi-estructurados	Datos no estructurados
Descripción	<ul style="list-style-type: none">* Representados con un formato estricto* Ejemplo: datos de una BD	<ul style="list-style-type: none">* Poseen cierta estructura, aunque no toda la información tiene necesariamente la misma estructura.* Información del esquema mezclada con los valores de los datos.* Self-describing data. Esquemas que por sí mismos describen la información que contienen. Por ejemplo JSON, o <u>XML</u>	Limitada información sobre los datos que contiene.

Características del XML:

- Documentos dan la estructura de los datos a través de tags HTML: `<un_tag> ... </un_tag>`.
- Un documento XML se puede pensar como un árbol. Los tags son los nodos, y los hijos de un nodo son los tags anidados. La raíz es el primer tag.
- Cómo es extensible (podemos definir nuestros propios tags) es utilizada para transmitir información (y no sólo almacenar documentos semi-estructurados) en diversas áreas.
- Se les puede definir un schema que restrinja / defina la información guardada.

Queries en XML:

- XPath: Consultas utilizando caminos en el árbol XML.
 - Ejemplo: /university-3/instructor/name
- XQuery
 - Usa sentencias como *for ... let ... where ... order by ...result ...*

Stream Databases

Definición: Son bases de datos que se ocupan de situaciones en las cuales los datos llegan en forma continua, por ejemplo:

- Los clicks que se hacen en las páginas web.
- Información proveniente de sensores (en tiempo real).

Formalmente, un stream, es una bolsa de pares $\langle s, t \rangle$ donde s es una tupla y t es el timestamp que denota la llegada lógica de la tupla al stream.

La diferencia fundamental entre un DBMS clásico y un DSMS (Data Stream Management System) es el modelo de datos que corresponde a los streams. En lugar de hacer los queries sobre datos que están en disco, los queries se hacen sobre datos que van ingresando y que están disponibles solo por un periodo muy corto de tiempo.

Base de datos espacial

Definición: Un sistema de base de datos espacial es un sistema de base de datos que cuenta con los tipos de datos espaciales (SDT) en su modelo de datos y lenguaje de consulta.

Tienen que poder sacar ventaja de los datos de “ubicación”. Los queries que resuelven incluyen los siguientes tipos:

- Objetos de un cierto tipo en un cierto rango.
- Vecino más cercano de un cierto tipo de objeto.
- Intersección o superposición entre 2 “objetos”.

Base de datos móviles

Definición: Una base de datos móvil es una base de datos que puede ser instalada en un dispositivo de computación móvil a través de una red móvil. El cliente y el servidor tienen conexiones inalámbricas. La memoria caché se mantiene para almacenar los datos frecuentes y transacciones de manera que no se pierdan debido a un fallo de conexión

El SMDB móvil debe replicar y mantener sincronizados los datos del servidor centralizado y los del dispositivo móvil.

RDF (Resource Description Format) Management Systems

Definición: Un dataset de tipo RDF tiene 2 tipos de datos:

- Explícitamente declarados,
- Implícitos, originados en restricciones semánticas. Estos datos se obtienen extendiendo los datos explícitos con las restricciones semánticas para convertirlos en explícitos.
 - Este proceso de extensión se llama *entailment*.

Ejemplo:

1. Explícitos:
 - a. Sócrates es humano
 - b. Los humanos son mortales
2. Implícito: El entailment concluye que Sócrates es mortal.

11. Data Mining

Definición: Es la extracción de patrones o información interesante (no trivial, implícita, previamente desconocida y potencialmente útil) de grandes bases de datos.

Aplicaciones:

- Asociación
 - Correlación y causalidad de hechos.
 - Por ejemplo, estudiar el nivel de estudios alcanzado de una persona según la categoría social de su familia.
- Clasificación y predicción
 - Generar modelos o funciones que sean capaces de clasificar elementos en clases.
 - Por ejemplo, predecir si una persona estará interesada en un crédito, según su edad y estatus financiero.
- Cluster analysis
 - Agrupar elementos para formar clases (clusters).
 - Por ejemplo, agrupar ads del mismo rubro.
- Análisis de outliers
 - Por ejemplo, para detectar fraudes o eventos raros.
- Análisis de tendencias y evolución

Técnicas

Supervisadas

Definición: Las técnicas supervisadas son aquellas donde los datos de entrenamiento están anotados con la respuesta esperada. Típicamente, la anotación es manual, y por lo tanto costosa.

Redes neuronales

La clase de problemas que mejor se resuelven con las redes neuronales son los mismos que el ser humano resuelve mejor pero a gran escala:

- Asociación
- Evaluación
- Reconocimiento de Patrones

Las redes neuronales son ideales para problemas que son muy difíciles de calcular

- No requieren de respuestas perfectas.
- Sólo respuestas rápidas y buenas.

Árboles de decisión

Idea: Clasificar objetos, en base a una serie de preguntas sobre sus atributos.

- Los nodos internos son preguntas sobre atributos.
- Las hojas representan las clases resultantes.

Regresión lineal

Definición: Técnica estadística para modelar e investigar la relación entre dos o más variables.

No supervisadas

Definición: Las técnicas no supervisadas tienen la tarea de inferir una función para describir la estructura oculta a partir de datos no etiquetados.

Clustering

Cluster: una colección de objetos que son:

- Similares dentro del cluster.
- Diferentes de los objetos en los otros clusters.

Clustering: Agrupar un conjunto de datos en clusters. Requiere definir la noción de similitud entre elementos. Se hace a través de una función distancia.

Un buen método de clustering produce clusters de alta calidad con:

- Alta similitud en la clase
- Baja similitud entre clases

Aplicaciones típicas:

- Como una herramienta independiente para tener una idea sobre la distribución de los datos
- Como un proceso previo a usar otros algoritmos

Formas de obtener un cluster

- Jerárquicas
 - Dos tipos:
 - Aglomerativo: Comienza con observaciones singulares y las une para formar elementos superiores en jerarquía.
 - Divisivo: Comienza con un grupo con todas las observaciones y se divide en subgrupos.
 - No hay decisión acerca del número de clusters.
 - Puede haber un criterio de parada como número de clusters máximo o distancia mínima para la unión de dos clusters.
 - Pueden ser muy lentos.
- No jerárquicas
 - Dado k , encontrar una partición de los elementos, en k clusters, que optimice cierto criterio de partición.
 - Son más rápidas y más fiables que los métodos jerárquicos.

Reglas de Asociación

Idea: Generar reglas del tipo IF condición THEN resultado, en forma automática.

12. Datawarehouse

Definición: Un datawarehouse es una colección de datos cuyo propósito es guiar la toma de decisiones de negocio (Business Intelligence). Proveen acceso para análisis complejo, descubrimiento de conocimiento y dan soporte a demandas de alta performance. Los datawarehouses cumplen las propiedades INTS:

- **Integrado:** Los datos son volcados al Datawarehouse desde diferentes fuentes e integrados en un todo consistente.
- **No volátil:** Los datos son estables. En general siempre se agregan datos pero no se quitan. Esto permite análisis retrospectivos sobre la marcha del negocio.
- **Time variant:** Todos los datos del datawarehouse refieren a un particular momento en el tiempo (como una “foto” o “snapshot”).
- **Subject oriented:** Los datos almacenados en el datawarehouse proveen información sobre un tema en particular en vez de atender la operatoria de gestión de la compañía.

Datos Operacionales vs Datawarehouse

	Datos Operacionales	Datawarehouse
Contenido	Valores elementates	Datos sumarizados, derivados
Organización	Por aplicación	Por tema
Estabilidad	Dinámicos	Estáticos hasta su actualización
Estructura	Optimizada para uso transaccional (NORMALIZADA)	Optimizada para queries complejas (DESNORMALIZADA)
Frecuencia de acceso	Alta	Media y baja
Tipo de acceso	Lectura / Escritura Actualización campo por campo	Lectura Sumarización
Uso	Predecible Repetitivo	Ad hoc Heurístico
Tiempo de respuesta	Segundos	Segundos a minutos
Cantidad de registros involucrados	A lo sumo decenas	Cientos - miles

Explotación del Datawarehouse / ETL

Definición de ETL (Extract, Transform, Load): Proceso por el cual se obtienen los datos de las distintas fuentes, se los limpian, se los convierte al formato apropiado y se los cargan en el DW.

Etapas:

1. Migración: Tomar los datos de los sistemas operacionales y llevarlos a un área de preparación (staging area).
 - No se debe mover datos innecesarios (control preventivo).
2. Limpieza:
 - Corregir, estandarizar y completar los datos.

- Identificar datos redundantes.
- Identificar outliers.
- Identificar valores perdidos.
- Ejemplos:
 - Eliminar transacciones con monto = 0 (promociones, regalos)
 - Eliminar transacciones anuladas (balance = 0).
 - Normalizar nombres de marcas de auto, de direcciones
- 3. Transformación: Son procesos destinados a adaptar los datos al modelo lógico del DW.
 - Agregar metadata a los datos. El objetivo es facilitar el acceso e interpretación posterior del contenido del DW.
 - Fuente
 - Descripción de operaciones de transformación
 - Desnormalización de los datos (el propósito es mejorar la performance).
 - Sumarizaciones. Muestran los datos de una manera más resumida. Permiten:
 - Acelerar los tiempos de análisis.
 - Ocultar la complejidad de los datos.
 - Proveer varias vistas del mismo conjunto de datos detallados (dimensiones en el “modelo dimensional” que se describe más adelante).
- 4. Carga: Carga de los datos en un soporte físico (por ejemplo una BD).
 - Métodos:
 - Full Refresh. Cargamos todos los datos de vuelta.
 - Incremental. Sólo cargamos los últimos datos.
- 5. Conciliación o validación:
 - Verificar integridad de los datos. Deben ser correctos y estar completos.
 - Métodos:
 - Completa, al final del proceso.
 - Por etapas, a medida que se cargan los datos.

Modelado de datos

¿De qué modo deben diseñarse las bases de datos que conforman un Datawarehouse para soportar eficientemente los requerimientos de los usuarios?

Técnicas

Modelo E - R

El modelo E - R (o Entity - Relationship) se basa en tres elementos: *entidades*, *relaciones*, *atributos*.

Las *entidades* pueden tener distintos *atributos*, y estas *entidades* se vinculan entre sí mediante *relaciones*.

Modelo Dimensional

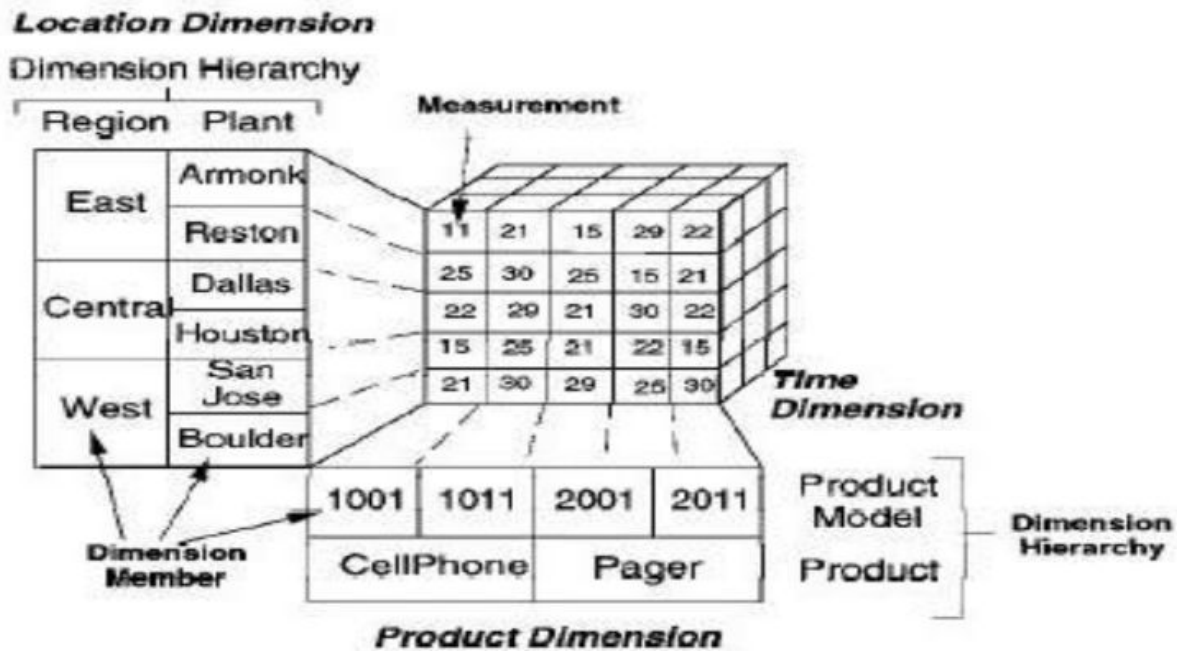
El modelo dimensional se basa en tres elementos: *hechos*, *dimensiones*, *medidas*.

- **Hechos:** Colección de ítems de datos. Cada hecho representa una transacción o un evento del negocio.
- **Dimensión:** Colección de miembros del mismo tipo (cosas relacionadas).
 - Cada hecho está conectado a una dimensión.
 - Determinan el contexto de los hechos.
 - Ejemplo: Tiempo es una dimensión de miembros (Meses, Trimestres, Años).

- Una dimensión puede estar formada por una jerarquía de miembros (o sea, no necesariamente es una lista de miembros).
- **Medida:** Es un valor numérico de un hecho que representa el desempeño de un hecho relativo a una dimensión.
 - Ejemplos: Ventas en \$\$, cantidad de productos, total de transacciones, etc.

Cubo OLAP

Definición: Cada dimensión del cubo es una dimensión del modelo. Cuando hay más de 3 dimensiones, se lo suele llamar *hipercubo*.

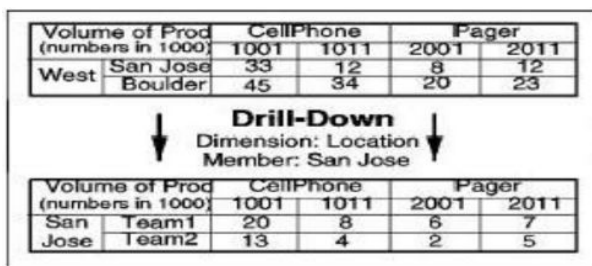


En este contexto, las dimensiones son una forma de clasificar los datos. Los datos son las medidas, y los definimos a partir de los hechos recabados.

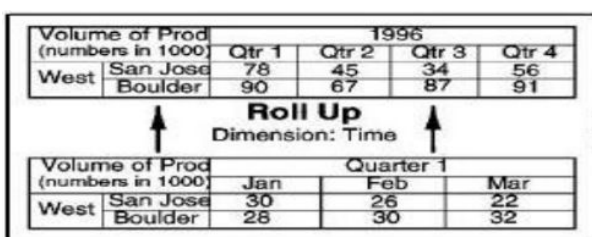
Operaciones

Nivel de granularidad:

- **Drill Down:** especialización de los datos al introducir una nueva dimensión a la vista o bajar en la jerarquía de dimensiones.

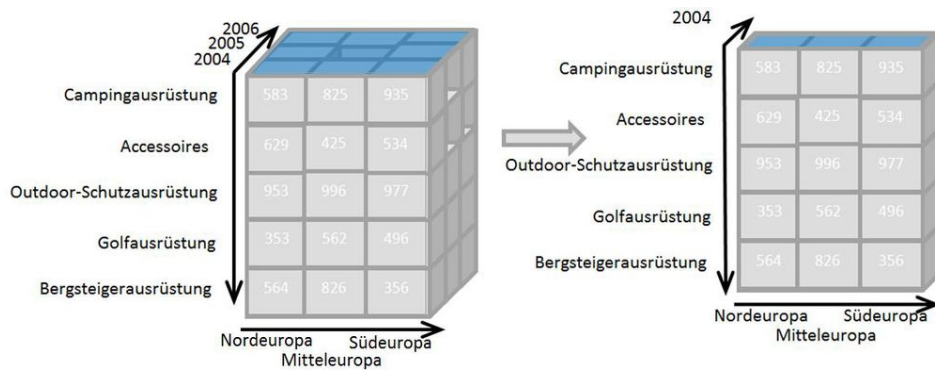


- **Roll Up:** agregación de datos de la vista al reducir dimensiones o escalar en la jerarquía de dimensiones.



Navegación por las dimensiones:

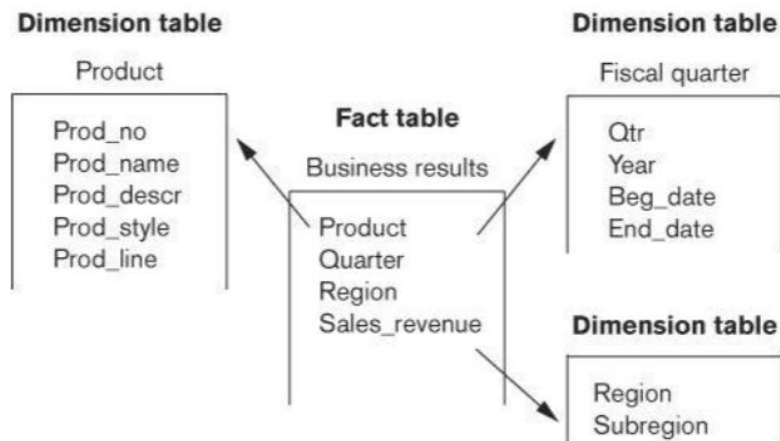
- Slice: es un subconjunto del “array multidimensional” que tiene un único valor para una o más dimensiones. Es una “rebanada” del cubo



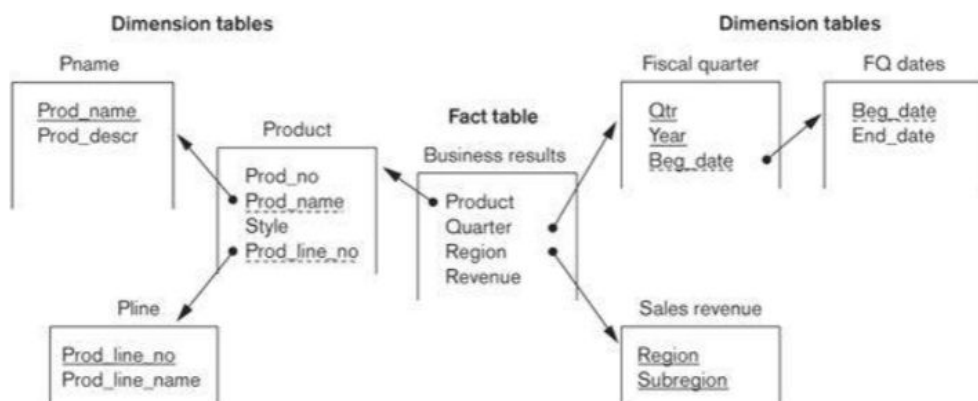
- Dice: es como el “slice” pero para 2 ó más valores de una o más dimensiones.

Modelos básicos dimensionales

- **Star:** Consiste en una tabla de hechos central y n tablas individuales para cada dimensión.



- **Snowflake:** Nuevamente hay una tabla central de hechos, pero las dimensiones están organizadas en forma jerárquica.



13. Long Duration Transactions

Definición: Una *Long Transaction* es una transacción que toma demasiado tiempo como para poder tomar un *lock* sobre un recurso compartido. Dependiendo de la situación, “demasiado” puede ser segundos, minutos o horas.

Saga: Una saga es una colección de acciones, que juntas forman long duration “transaction”. Una saga consiste de:

1. Una colección de acciones.
2. Un grafo dirigido cuyos nodos son las acciones o los nodos terminales *Abort* o *Complete*. Ningún eje sale de los nodos terminales.
3. Una indicación de cuál nodo es el inicial.

Los caminos del grafo representan las posibles secuencias de acciones. Los caminos que llevan al nodo *Abort* representan las secuencias que harían que la saga sea deshecha. Los caminos que llevan a *Complete* representan las secuencias de acciones que harían que las acciones realizadas permanezcan en la base de datos.

El control de concurrencia de las sagas se puede separar en dos partes:

1. Cada acción se puede considerar como una transacción “corta” que puede ejecutarse utilizando mecanismos de control de concurrencia convencionales, como *locking*.
2. La transacción en general se maneja mediante transacciones compensatorias o inversas.
 - Cada acción A de una saga tiene una transacción inversa A^{-1} . Intuitivamente, si se ejecuta A y después A^{-1} , el estado resultante de la base será el mismo como si ninguna se hubiese ejecutado.
 - Si la ejecución de una saga lleva al nodo *Abort*, se deshacerá la misma ejecutando las transacciones inversas en el orden inverso a como se ejecutaron originalmente.

Ejemplo:

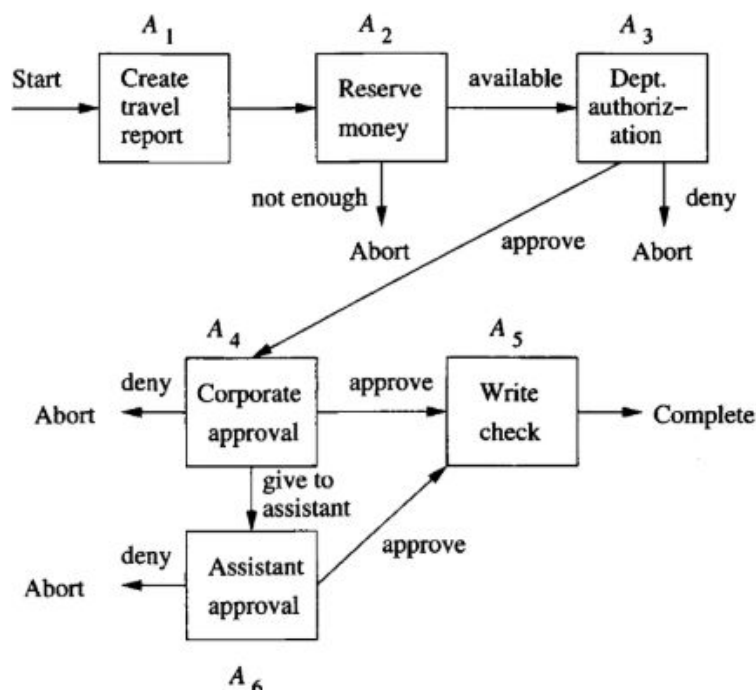


Figure 19.11: Workflow for a traveler requesting expense reimbursement

14. Open Data

Definición: Se denomina Open Data a la puesta en disponibilidad pública por parte de los estados de datos en forma digital a través de Internet de manera que permita y promueva su análisis y reutilización.

El acceso a datos gubernamentales se considera abierto si los datos son puestos a disposición del público cumpliendo con los siguientes principios:

1. **Completo:** Toda los datos están disponibles, sin ningún tipo de limitación de seguridad o privilegio.
2. **De primera fuente:** Los datos son coleccionados desde la fuente, con el mayor grado posible de granularidad, sin ningún tipo de modificación o agregación.
3. **En tiempo:** Los datos están disponibles tan pronto como sea necesario para preservar el valor de los datos.
4. **Accesible:** Los datos están disponibles para el rango más amplio de usuarios para el rango más amplio de propósitos.
5. **Procesables por computadoras:** Los datos están estructurados razonablemente como para permitir su procesamiento automático.
6. **No discriminatorios:** Los datos están disponibles para todos, sin necesidad de registro. El acceso tiene que poder ser accedido de forma anónima (incluso a través de proxies anónimos).
7. **No propietario:** Los datos tienen que tener un formato del que ninguna entidad tiene control exclusivo.
8. **Sin licencia:** Los datos no tienen que estar sujetos a derechos de autor, patente, etc.

15. Administrador de Datos

Diferencia entre DBA y administrador de datos

- Un DBA es un especialista en un motor de BD. Sabe manejar sus DDL, DML y DCL.
- Un administrador de datos es un especialista en los datos de una organización.

Tareas de un administrador de datos

Diseño lógico

- Analizar requerimientos.
- Realizar modelado a partir de los requerimientos.
- Definir estándares (nombres de entidades, relaciones, etc.) y asegurar su cumplimiento.

Diseño físico

- Asistir al DBA en la creación de modelos físicos a partir de los modelos lógicos.
- Analizar el volumen de datos y requerimientos de espacio.
- Ejecutar backups y recoveries.
- Verificar integridad de los datos en las BD.

Bibliografía

Este apunte fue basado en:

- Las clases de Cecilia
- Las diapositivas encontradas en la página del DC
- Otros apuntes de la materia